1998-03

Single-frequency measurements using undersampling methods

Chia, Eng S.

Monterey, California. Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

SINGLE-FREQUENCY MEASUREMENTS USING UNDERSAMPLING METHODS

by

Eng S. Chia

March 1998

Thesis Advisor: Phillip E. Pace

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE March 1998	3. REPORT TYPE AND DATES COVERED Master's Thesis

4. TITLE AND SUBTITLE
SINGLE-FREQUENCY MEASUREMENTS USING UNDERSAMPLING METHODS

5. FUNDING NUMBERS

6. AUTHOR(S)
Chia, Eng S.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Research and Development Office, Washington DC

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.
MATLAB is a registered trademark of the MathWorks, Inc.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*

The objective of this study is to verify the Symmetrical Number System (SNS) undersampling receiver architecture using software and to investigate implementation issues using digital signal processing (DSP) hardware. In the software design, a MATLAB program is written to determine a single sinusoidal input frequency using this receiver architecture. Each channel of the SNS undersampling receiver consists of a low speed ADC, a discrete Fourier transform followed by a constant threshold device to detect the signal's frequency bin. The detected frequency bins are then recombined in a SNS-to-decimal algorithm to recover the frequency of the signal. Error rate performance in a Gaussian noise environment at the input stage is evaluated. In the hardware design, a sinusoidal waveform is digitized, discrete Fourier transformed and converted from the SNS format to a decimal value using a single channel digital signal processor. Implementation difficulties and design issues are discussed.

14. SUBJECT TERMS
Symmetrical Number System, Symmetrical folding, Undersampling, Discrete Fourier Transform.

15. NUMBER OF PAGES
96

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

SINGLE-FREQUENCY MEASUREMENTS USING UNDERSAMPLING METHODS

Eng S. Chia
Major, Republic of Singapore Airforce
B.S., National University of Singapore, 1989

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
March 1998

ABSTRACT

The objective of this study is to verify the Symmetrical Number System (SNS) undersampling receiver architecture using software and investigate implementation issues using Digital Signal Processing (DSP) hardware. In the software design, a MATLAB program is written to determine a single sinusoidal input frequency using this receiver architecture. Each channel of the SNS undersampling receiver consists of a low speed ADC, a discrete Fourier transform followed by a constant threshold device to detect the signal's frequency bin. The detected frequency bins are then recombined in a SNS-to-decimal algorithm to recover the frequency of the signal. Error rate performance in a Gaussian noise environment at the input stage is evaluated. In the hardware design, a sinusoidal waveform is digitized, discrete Fourier transformed and converted from the SNS format to a decimal value using a single channel digital signal processor. Implementation difficulties and design issues are discussed.

TABLE OF CONTENTS

I. INTRODUCTION

A. UNDERSAMPLING

The digitization of a signal is usually governed by the Nyquist theorem where the sampling frequency is at least twice the signal bandwidth. The Nyquist theorem however, places a limitation only on the information that can be derived from a single set of digitized data [Ref. 1]. If the sampling frequency is less than twice the bandwidth of the signal being digitized, aliasing and consequently ambiguities occur. With additional information however, ambiguous frequency components due to undersampling may be resolved. Such information may come from, for example, trial sampling periods. Rader [Ref. 2] described how trial sampling periods can be used to recover periodic signals. The trial sampling period which yields the waveform of smallest variation is considered to be the correct period and the resulting waveform the correct waveform.

Pace, Leino and Styer [Ref. 3] examined the relationship between the Discrete Fourier Transform (DFT) and the Symmetrical Number System (SNS) as a means of resolving

single frequency undersampling aliases. They showed that the DFT encodes the frequency information of a signal in a format that is in the same form as the SNS. In addition, they proved analytically that aliases resulting from undersampling a single-frequency signal could be resolved using 2 or more channels. Each channel in a SNS undersampling receiver contains a low speed ADC, a DFT and a threshold device to detect the input signal bin number in the frequency domain. The bin numbers from each channel are then recombined to resolve the signal's frequency.

B. PRINCIPAL CONTRIBUTIONS

First, this thesis verifies the SNS undersampling theory advanced by Pace, Leino and Styer [Ref. 3]. An algorithm is written and coded in MATLAB to prove the methodology and to show that the frequency of an undersampled signal can be accurately measured. The algorithm is also simulated in a Gaussian noise environment. Error rates for the different noise levels are obtained as a function of the signal to noise ratio. Since the Fast Fourier Transform (FFT) is not suitable for computing DFTs in this application, alternative methods are suggested for real-time applications.

Second, possible hardware implementation problems are investigated based on a Digital Signal Processing (DSP) platform. Several problems were encountered: the need for stable sampling frequencies, large memories and alternative methods for computing DFT for fast response time. Integration into future EW receivers must take these factors into consideration.

Undersampling offers several advantages [Ref. 4]. It allows the resolution of very high frequencies in EW receivers using low speed ADCs. This is especially so if several SNS channels are used. In particular, the use of undersampling in the design of receivers will reduce their cost and complexity.

C. THESIS ORGANIZATION

In Chapter II, the relationship between the SNS and the digital frequency domain as mapped by the DFT is examined as a means of resolving single-frequency undersampling ambiguities. It shows how the frequency of a signal that is undersampled at two different sampling frequencies (two-channel) can be determined. In order to use lower sampling frequencies, the two-channel case can be extended to three or

more channels. In particular the three-channel case is discussed.

In Chapter III, algorithms for the two-channel and three-channel receivers are developed and coded in MATLAB to measure the frequency of an incoming signal. Each section of the software is explained in detail. Results are obtained based on different Gaussian noise levels.

A feasibility study/design for the two-channel case is carried out in Chapter IV using a DSP development kit. The suitability of using a DSP platform and its associated problems are discussed.

Chapter V states some conclusions and recommendations for future research.

II. BACKGROUND INFORMATION

A. INTRODUCTION

Digitization of a signal is usually governed by the Nyquist criterion when the input signal is bandlimited to $0 < f < f_s/2$ where f_s is the sampling frequency. For higher frequencies (i.e. $f > f_s/2$), the process of undersampling gives rise to ambiguities. However, with additional information (or channels), the frequency components $f > f_s/2$ can be resolved.

Pace, Ramamoorthy and Styer [Ref. 5] showed that the discrete Fourier transform (DFT) naturally encodes the frequency information of a signal in the same format as the symmetrical number system (SNS). Consequently, aliases from undersampling can be resolved using this method. The theory set forth is elaborated in [Ref. 3].

B. DISCRETE FOURIER TRANSFORM (DFT)

Since all signals consist of sinusoids, for simplicity, a single frequency sinusoidal waveform is used for analysis. Assume the sinusoidal signal is

$$x(t) = 2\cos \omega t$$

(1)

and after sampling

$$x(n) = 2\cos \omega n.$$

(2)

The DFT of x(n) is given by [Ref. 6]:

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-j(2\pi nk/N)} \qquad k = 0, 1, \ldots, N-1.$$

(3)

Applying the DFT to x(n) results in a discrete spectrum where $|X(k)|^2$ is the energy contained in the signal at each digital frequency $\omega = 2\pi k/N$. The spectrum X(k) has N indices with the digital frequency of each index given by:

$$\left[0, 2\pi\frac{1}{N}, \ldots, 2\pi\frac{(N/2)}{N}, 2\pi\frac{(N/2+1)}{N}, \ldots, 2\pi\frac{(N-2)}{N}, 2\pi\frac{(N-1)}{N}\right] \quad \text{for N even}$$

(4)

and

$$\left[0, 2\pi\frac{1}{N}, \ldots, 2\pi\frac{(N-1)/2}{N}, 2\pi\frac{(N+1)/2}{N}, \ldots, 2\pi\frac{(N-2)}{N}, 2\pi\frac{(N-1)}{N}\right] \quad \text{for N odd.}$$

(5)

The analog frequency corresponding to each index is obtained by multiplying each value by f_s. Since signals with digital frequencies in the range $\pi < \omega < 2\pi$ are indistinguishable from signals with digital frequencies $0 < \omega < \pi$, the digital frequency of each index can also be written as:

$$\left[0, 2\pi\frac{1}{N}, \ldots, 2\pi\frac{(N/2)}{N}, 2\pi\frac{(N/2-1)}{N}, \ldots, 2\pi\frac{2}{N}, 2\pi\frac{1}{N}\right] \quad \text{for N even} \tag{6}$$

and

$$\left[0, 2\pi\frac{1}{N}, \ldots, 2\pi\frac{\lfloor N/2 \rfloor}{N}, 2\pi\frac{\lfloor N/2 \rfloor}{N}, \ldots, 2\pi\frac{2}{N}, 2\pi\frac{1}{N}\right] \quad \text{for N odd.} \tag{7}$$

where $\lfloor x \rfloor$ is the floor function and represents the greatest integer less than or equal to x. Thus the spectrum X(k) resolves into N integer indices and incoming signals will map into unique bins:

$$\left[0, 1, \ldots, \frac{N}{2}, \frac{N}{2}-1, \ldots, 2, 1\right] \quad \text{for N even,} \tag{8}$$

$$\left[0, 1, \ldots, \left\lfloor\frac{N}{2}\right\rfloor, \left\lfloor\frac{N}{2}\right\rfloor, \ldots, 2, 1\right] \quad \text{for N odd.} \tag{9}$$

For example, for N = 5 (f_s = 5 Hz and the sampling duration T_1 is 1 second), the output bins after the DFT are [0 1 2 2 1] for input frequencies of [0 1 2 3 4] Hz. These DFT bins are repeated for higher frequencies as illustrated in Figure 1. In this figure the abscissa corresponds to the incoming frequency and the ordinate corresponds to the bin into which the signal is resolved.

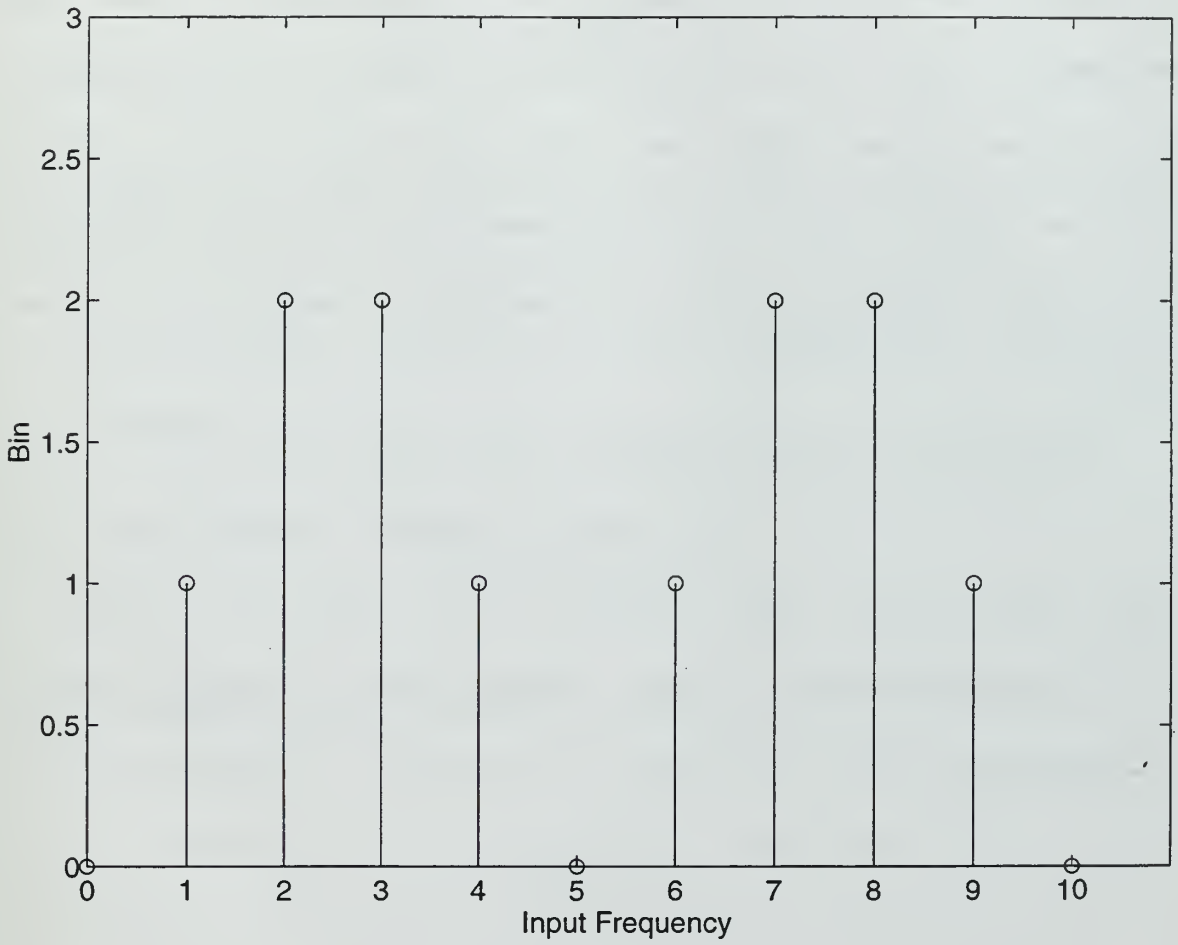


Figure 1: DFT bin mapping for input frequencies f= 0 to 10 for N = 5 (f_s = 5 Hz sampling for 1 second).

C. THE SYMMETRICAL NUMBER SYSTEM (SNS)

The SNS is composed of a number of pairwise relatively prime (PRP) moduli. The integers within each SNS modulus however, are derived from a symmetrically folded waveform. The symmetrically folded waveform corresponding to each SNS PRP moduli (m_i), has a folding period equal to the modulus. The integer values within each SNS modulus are derived from a mid-level quantization of the symmetrical folding waveform. The formal definition of a symmetrical residue is given below:

Definition: For an integer h such that $0 \leq h < m$

$$x_h = \min\{h, m-h\} \tag{10}$$

If this function is extended periodically with period m, that is,

$$x_{h+nm} = x_h \tag{11}$$

where $n \in \{0, \pm 1, \pm 2, \ldots\}$ then x_h is called a symmetrical residue of (h+nm) modulo m. For m even, let x be the row vector

$$x = \left[0, 1, \ldots, \frac{m}{2}, \frac{m}{2} - 1, \ldots, 2, 1 \right]. \tag{12}$$

For m odd, let x be the row vector

$$x = \left[0, 1, \ldots, \left\lfloor \frac{m}{2} \right\rfloor, \left\lfloor \frac{m}{2} \right\rfloor, \ldots, 2, 1 \right]. \tag{13}$$

where $\lfloor x \rfloor$ again represents the floor function resulting in the greatest integer less than or equal to x. These two vectors consist of the symmetrical remainder elements x_h, $0 \leq h < m$.

D. RELATIONSHIP BETWEEN DFT AND SNS

From the above, it is obvious that the DFT maps real signals naturally into the SNS. That is, in Section C, if we let the modulus m represent the sampling frequency multiplied by the sampling time (i.e., $f_s T_1$), then equations (12) and (13) are in the same form as equations (8) and (9) where $N = f_s T_1$. Thus the SNS provides a convenient framework for undersamping signal analysis.

Table 1 displays the input frequencies and the resulting DFT bins for sampling frequencies 5 Hz and 6 Hz respectively.

Input Frequency f	DFT Bins	
	f_s= 5 Hz	f_s= 6 Hz
0	0	0
1	1	1
2	2	2
3	2	3
4	1	2
5	0	1
6	1	0
7	2	1
8	2	2

Table 1: Input Frequency and Resulting DFT Bins for 2 Channel Example.

The frequencies are resolved as described in equations (12) and (13). By considering two or more channels, it is possible to unambiguously resolve the signal frequencies in the dynamic range determined by the SNS. One method is to devise a look-up table similar to that shown in Table 1. However this method is inefficient for high frequencies; large memories are required. An alternative method is described below:

Suppose there are r channels and the incoming frequency is within the dynamic range of the system. To carry out the SNS-to-decimal conversion, we need to solve $f \equiv a_i \pmod{m_i}$

for i = 1,2,...r, where a_i is the corresponding detected DFT bin for each m_i. The Chinese Remainder Theorem states that there is a unique solution modulo $M=m_1*m_2 \ldots *m_r$. A standard method of solution is to find integers b_i such that $M*b_i/m_i \equiv 1 (\text{mod } m_i)$ where i = 1,2,....r in which case the solution is $f \equiv M*b_1*a_1/m_1 + M*b_2*a_2/m_2 + \ldots + M*b_r*a_r/m_r \ (\text{mod } M)$. In Sections F and G below, examples are given to illustrate this calculation.

E. DYNAMIC RANGE OF THE SNS

Let $m_1,\ldots, m_r$ be r pairwise relatively prime moduli, then the dynamic range, D (0:D-1) of a SNS system is given as follows:

- If all the moduli are odd, then the dynamic range of the system is

$$D = \min\left\{\frac{1}{2}\prod_{l=1}^{j} m_{i_l} + \frac{1}{2}\prod_{l=j+1}^{r} m_{i_l}\right\} \tag{14}$$

where j ranges from 1 to r-1 and $m_{i_2}, m_{i_3} \ldots m_{i_r}$ range over all permutations of {1,2,3,…,r}. For example, for a two-channel case with $m_1 = 5$, $m_2 = 7$,

$$D = \min\left\{\frac{m_1}{2}+\frac{m_2}{2}\right\}$$

or $D = 6.$

For a three-channel case with $m_1 = 3$, $m_2 = 5$, $m_3 = 7$,

$$D = \frac{1}{2}\min\left\{m_1 + m_2 m_3, m_2 + m_1 m_3, m_3 + m_1 m_2\right\}$$

or $D = 22.$

- If one of the moduli (m_1) is even, then the dynamic range of the system is

$$D = \min\left\{\frac{m_1}{2}\prod_{l=2}^{j} m_{i_l}+\prod_{l=j+1}^{r} m_{i_l}\right\} \qquad (15)$$

where j ranges from 1 to r-1 and $m_{i_2}, m_{i_3} \ldots m_{i_r}$ range over all permutations of $\{2,3,\ldots,r\}$. For example, for a two-channel case with $m_1 = 6$ $m_2 = 5$,

$$D = \min\left\{\frac{m_1}{2}+m_2\right\}$$

or $D = 8.$

For a three-channel case with $m_1 = 8$, $m_2 = 5$, $m_3 = 7$,

$$D = \min\left\{\frac{m_1}{2} + m_2 m_3, \frac{m_1}{2} m_2 + m_3, \frac{m_1}{2} m_3 + m_2\right\},$$

or $D = 27.$

Clearly, the dynamic range of an SNS system with one even modulus is superior to that using all odd moduli. Moreover, the greater the number of channels, the greater the dynamic range.

F. THE TWO-CHANNEL CASE

Figure 2 shows the block diagram of a two-channel receiver architecture to determine a single frequency f. In this architecture the ADC sampling frequencies f_{s1} and f_{s2} are relatively prime and $T_1 = 1$. The DFT outputs are thresholded to detect the frequency bins of the signal. The detected frequency bins a_1 and a_2 are then used by the SNS-to-decimal algorithm to determine the frequency of the input signal.

frequency bins a_1 and a_2 are then used by the SNS-to-decimal algorithm to determine the frequency of the input signal.

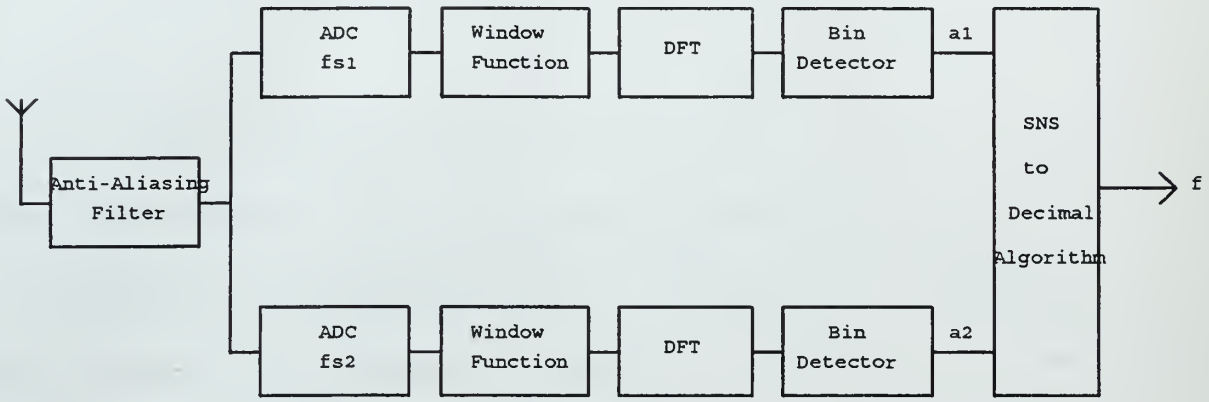


Figure 2: Block Diagram of a Two Channel Receiver Architecture.

Let $m_1 = f_{s1}$ and $m_2 = f_{s2}$ and suppose that the incoming frequency is within the dynamic range of the system. From Section D, we need to solve $f \equiv a_1 (\text{mod } m_1)$ and $f \equiv a_2 (\text{mod } m_2)$. The two congruence equations, $f \equiv a_1 (\text{mod } m_1)$ and $f \equiv a_2 (\text{mod } m_2)$ are solvable only if the greatest common divisor of m_1 and m_2 divides $(a_2 - a_1)$, a generalization of the Chinese Remainder Theorem [Ref. 7]. To solve for f, the diophantine equation

$$p*m_1+q*m_2 = (a_2 - a_1) \qquad (16)$$

must be solved for p and f is then calculated from the equation

$$f = a_1 + p*m_1. \qquad (17)$$

The code for this algorithm is shown in Appendix A.

For example, for sampling frequencies 5 and 6, m_1 and m_2 have values of 5 and 6 respectively ($T_1 = 1$). If the signal is resolved into bins a_1 (= 2) and a_2 (= 1) after the DFT, p is found to have a value of 1 and q is found have a value of -1. Thus, the input frequency from (17) is 2+1*5 = 7. This can also be verified as shown in Table 1.

G. THE THREE-CHANNEL CASE

Figure 3 shows the block diagram of a three-channel receiver architecture to determine a single frequency f. Similar to the two-channel case, the ADC sampling frequencies f_{s1}, f_{s2}, and f_{s3} are pairwise relatively prime and $T_1 = 1$. The DFT outputs are thresholded to detect the frequency bins of the signal. The frequency bins a_1, a_2 and a_3 are then used by the SNS-to-decimal algorithm to determine the frequency of the input signal.

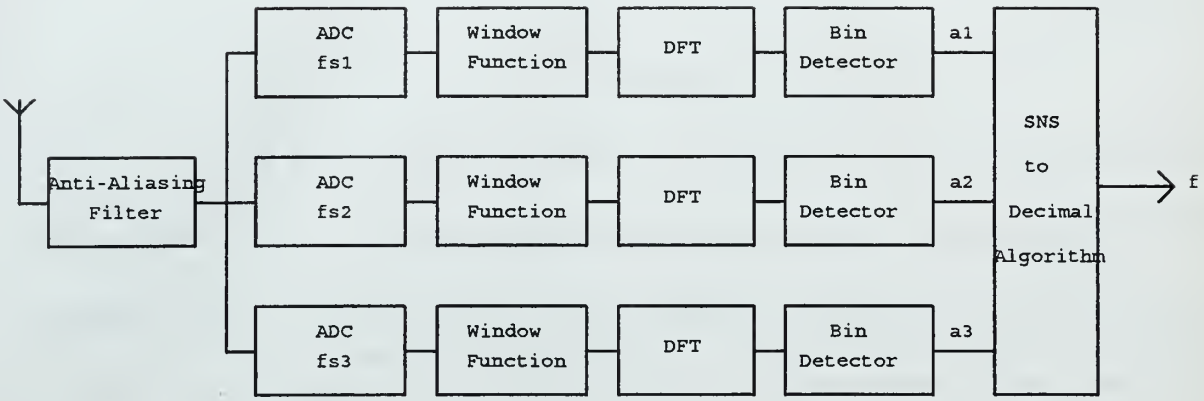


Figure 3: Block Diagram of a Three Channel Receiver Architecture.

In the three-channel solution, let $m_1 = f_{s1}$, $m_2 = f_{s2}$ and $m_3 = f_{s3}$ and suppose that the incoming frequency is within the dynamic range of the system. We need to solve $f \equiv a_1 (\text{mod } m_1)$ and $f \equiv a_2 (\text{mod } m_2)$ and $f \equiv a_3 (\text{mod } m_3)$. Using the Chinese Remainder Theorem and the Euclidean algorithm, the method of solution is to find integers b_i such that $M*b_i/m_i \equiv 1 (\text{mod } m_i)$ where i = 1,2,and 3 and $M = m_1 * m_2 * m_3$. The solution is then $f \equiv \pm M*b_1*a_1/m_1 \pm M*b_2*a_2/m_2 \pm M*b_3*a_3/m_3 \text{ (mod M)}$ where f is the frequency which falls within the dynamic range D of the system.

For example, let $m_1 = 5$, $m_2 = 6$ and $m_3 = 7$, so that M = 210 and D = 22. Suppose that the signal is resolved into bins a_1 (= 1), a_2 (= 2) and a_3 (= 2) after the DFT. For the three-channel case the b_i values must be found. Here, b_1, b_2 and b_3 are found to be -2, -1, and -3 respectively. Thus $f \equiv \pm 210(-2)(1)/5 \pm 210(-1)(2)/6 \pm 210(-3)(2)/7 \bmod(210)$ and we must choose the solution that falls within the SNS dynamic range D = 22 [0:21]. The correct combination $f \equiv 84 - 70 + 180 \bmod(210) \equiv 194 \bmod(210)$. Although 194 is out of the dynamic range, 210 - 194 = 16 is in the dynamic range so that f = 16 is the correct frequency.

H. NOISE CONSIDERATIONS

For a sinusoidal waveform, the Signal to Noise Ratio (SNR) is defined as

$$SNR = \frac{P}{2\sigma^2} \quad (18)$$

where P is the power of the signal and σ^2 is the noise power. Assuming a signal power of one, the noise power and amplitude are given by

$$\sigma^2 = \frac{1}{2\,SNR} \tag{19}$$

$$\sigma = \frac{1}{\sqrt{2\,SNR.}} \tag{20}$$

This σ is multiplied by a normally distributed random number sequence of zero mean and unit variance and added to the input signal as noise. The simulation results are given in Chapter III.

III. SOFTWARE DESIGN AND RESULTS

A. TWO-CHANNEL ALGORITHM

The two-channel case was described in Chapter II. An algorithm was constructed based on Figure 2. The software given in Appendix A can be divided into the following sections:

- Initialization. This section obtains all the parameters (number of iterations, input frequency, sampling frequencies, quantization levels) required.
- Iteration loop. This section consists of a loop (with an initial count of zero) to count the number of errors.
- Creation of Waveform. Based on the input frequency, a sinusoidal waveform is created with noise added.
- Sampling and Quantization. The waveform is then sampled at two different frequencies and quantized using a 14-bit ADC.
- Windowing. A rectangular window operation of width N = f_s * T_i = f_s (the total sampling/integration time is taken to be one)is carried out.

- DFT Operation. A DFT is then carried out on each sample, taking only the first half of the DFT output. The formula used for the DFT process is a simple pair of nested loops.
- Bin Detection. A non-adaptive (constant) threshold bin detector is then used to find the bin with the maximum value for each DFT output.
- SNS-to-Decimal Algorithm. The SNS-to-decimal algorithm as described in Chapter II is then used to calculate the incoming frequency.

A flow diagram of this algorithm is illustrated in Figure 4. The MATLAB code can be found in Appendix A.

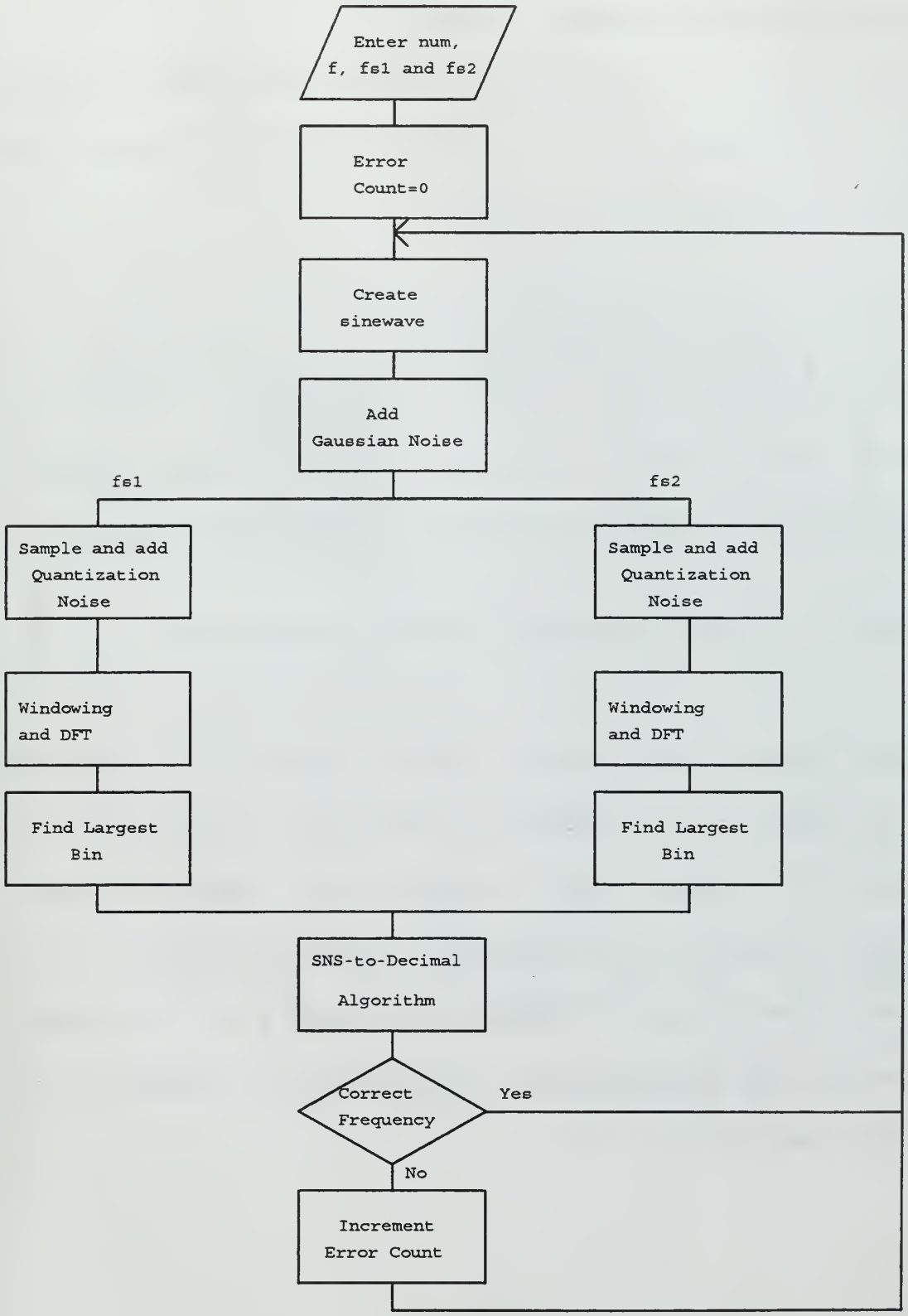


Figure 4: Two Channel Algorithm

B. TESTING OF TWO-CHANNEL SYSTEM

To test the two-channel case (sinusoidal signal without noise), the program is run with the following input and sampling frequencies shown in Table 2.

f	f_{s1}	f_{s2}	Dynamic Range	Remarks
7	5	8	0:8	Low input frequency
100	97	98	0:145	Consecutive sampling frequencies
1040	547	1200	0:1146	Sampling frequencies far apart
12125	12671	12919	0:12794	High input frequency

Table 2: Tested Input and Sampling Frequencies.

For example, with input signal frequency at 7 Hz as shown in Figure 5, the sampled signals at 5 Hz and at 8 Hz are shown in Figures 6 and 7 respectively. The DFT output for the two samples are shown in Figures 8 and 9. The resultant bins of the first halves of Figure 8 and 9 are then supplied to the SNS-to-decimal algorithm to be converted to the input frequency of 7 Hz.

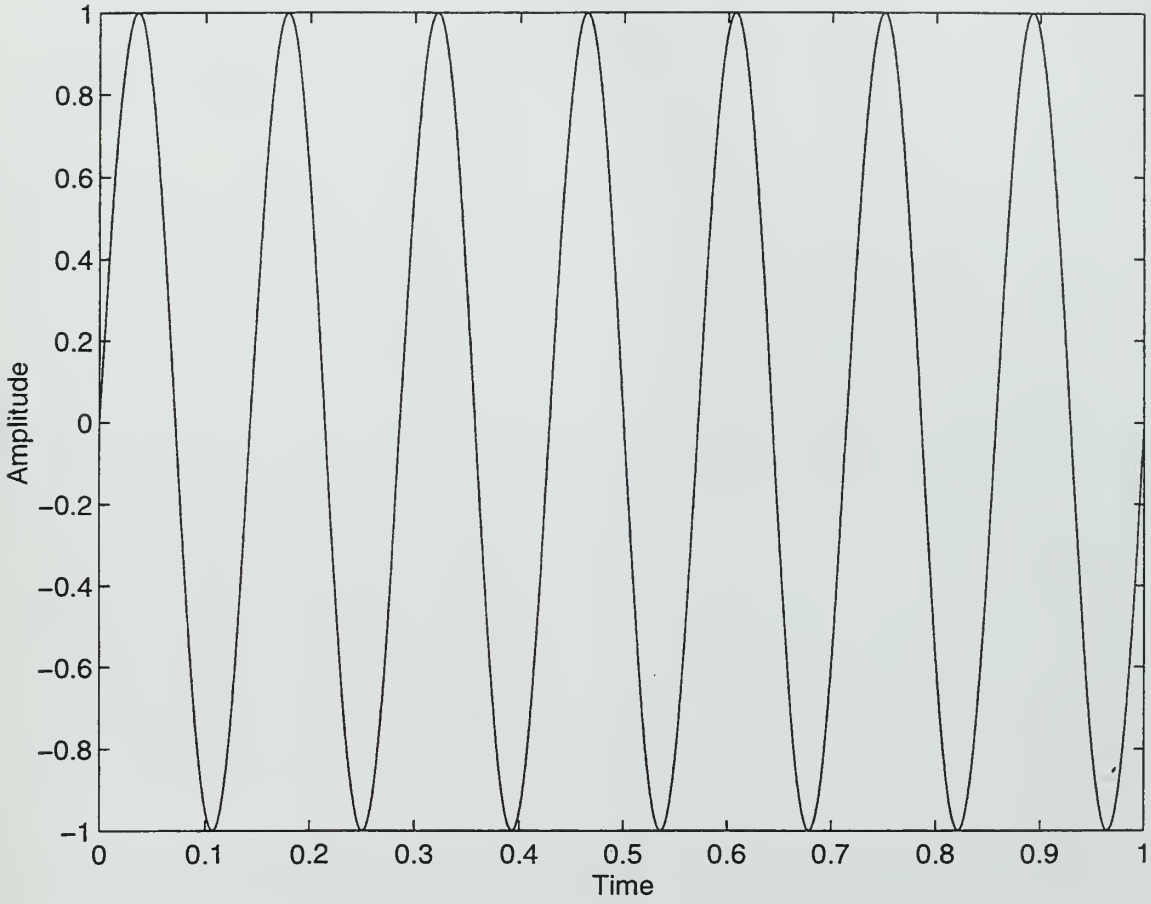


Figure 5: Input signal with frequency of 7 Hz.

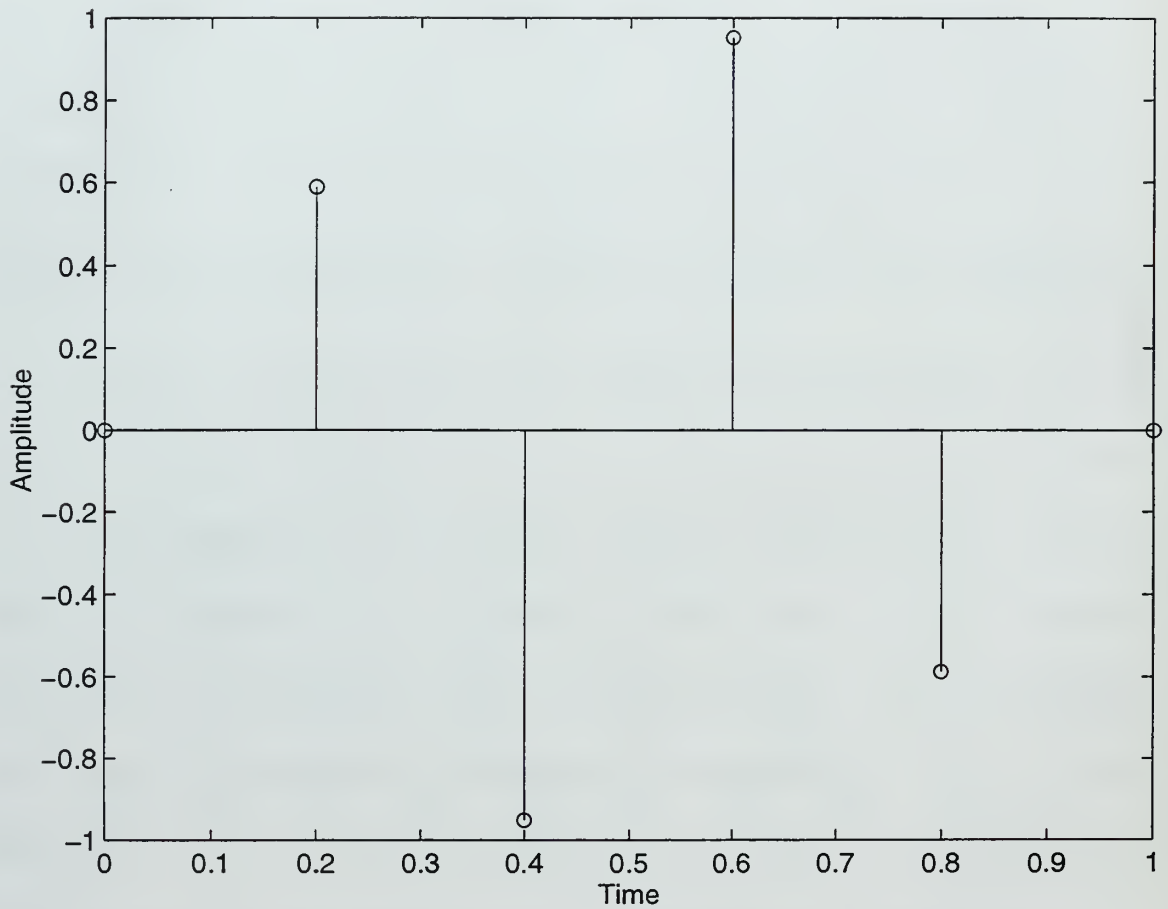


Figure 6: Sampled signal at frequency 5 Hz.

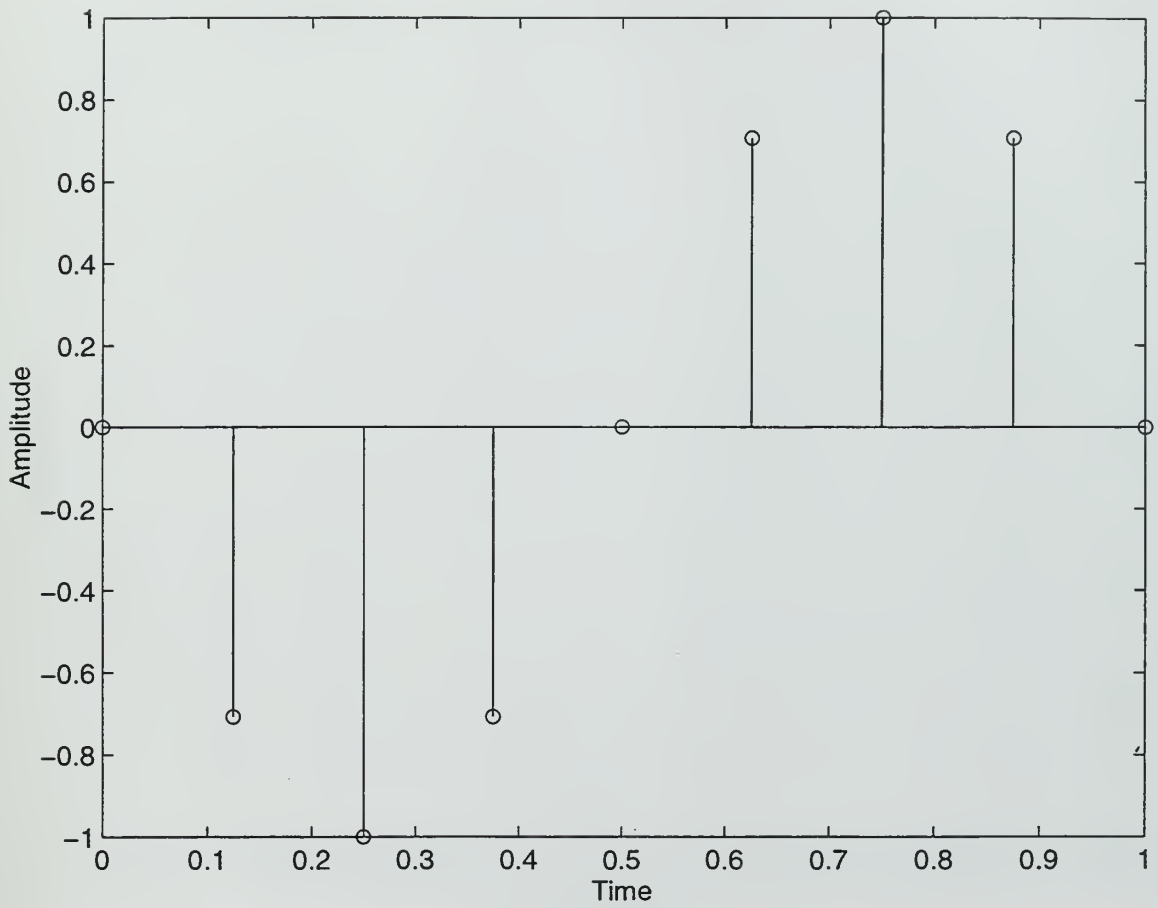


Figure 7: Sampled signal at frequency 8 Hz.

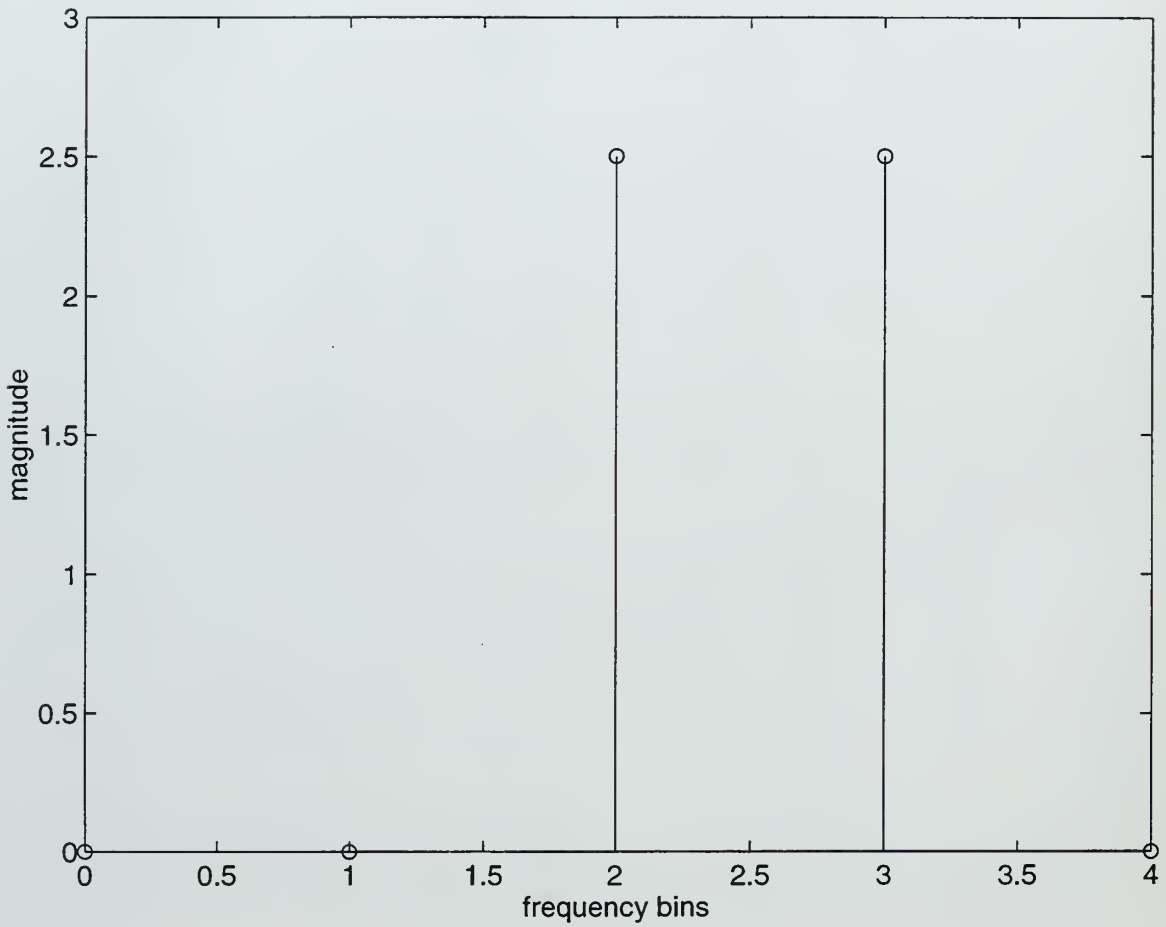


Figure 8: DFT output with fs1=5 Hz.

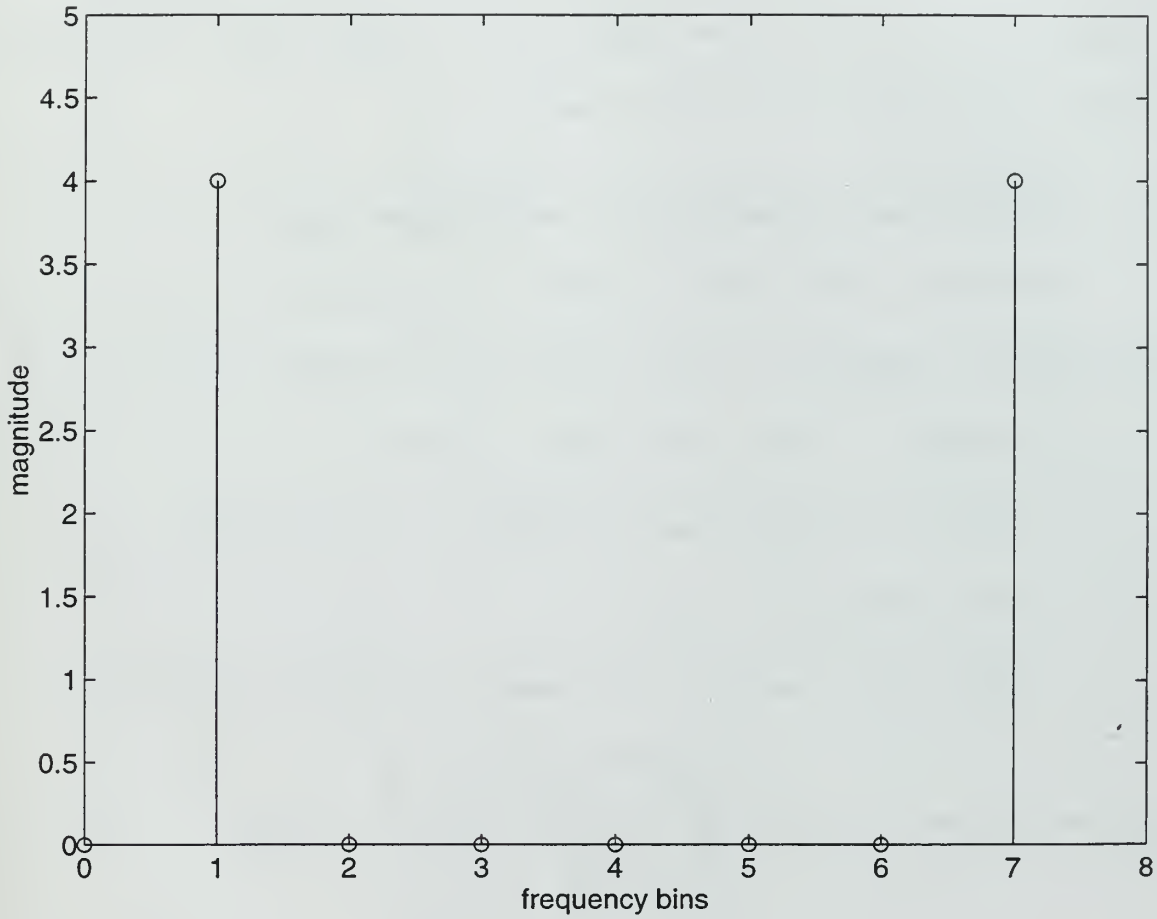


Figure 9: DFT output with fs2=8 Hz.

It is found that if one of the sampling frequencies was the same as the input frequency, the algorithm failed. This is because the resulting samples due to the same sampling frequency will consist of zeros. This problem can be solved by using at least two sets of sampling frequencies. Apart from this, the algorithm works well in this noise-free (high signal-to-noise ratio) environment.

C. SIMULATION PARAMETERS FOR TWO-CHANNEL CASE

To obtain the error rates in a noisy environment, the two-channel software is run with the following parameters:

- Number of iterations, num = 10000
- Signal to Noise Ratio, SNRDB = -30 to 30 dB
- ADC resolution, bit = 14
- Input and sampling frequencies as shown in Table 3.

f	f_{s1}	f_{s2}
9	10	11
90	91	92
900	901	902
9000	9001	9002

Table 3: Input and Sampling Frequencies.

D. RESULTS FOR TWO-CHANNEL CASE

The results obtained are shown in Figure 10.

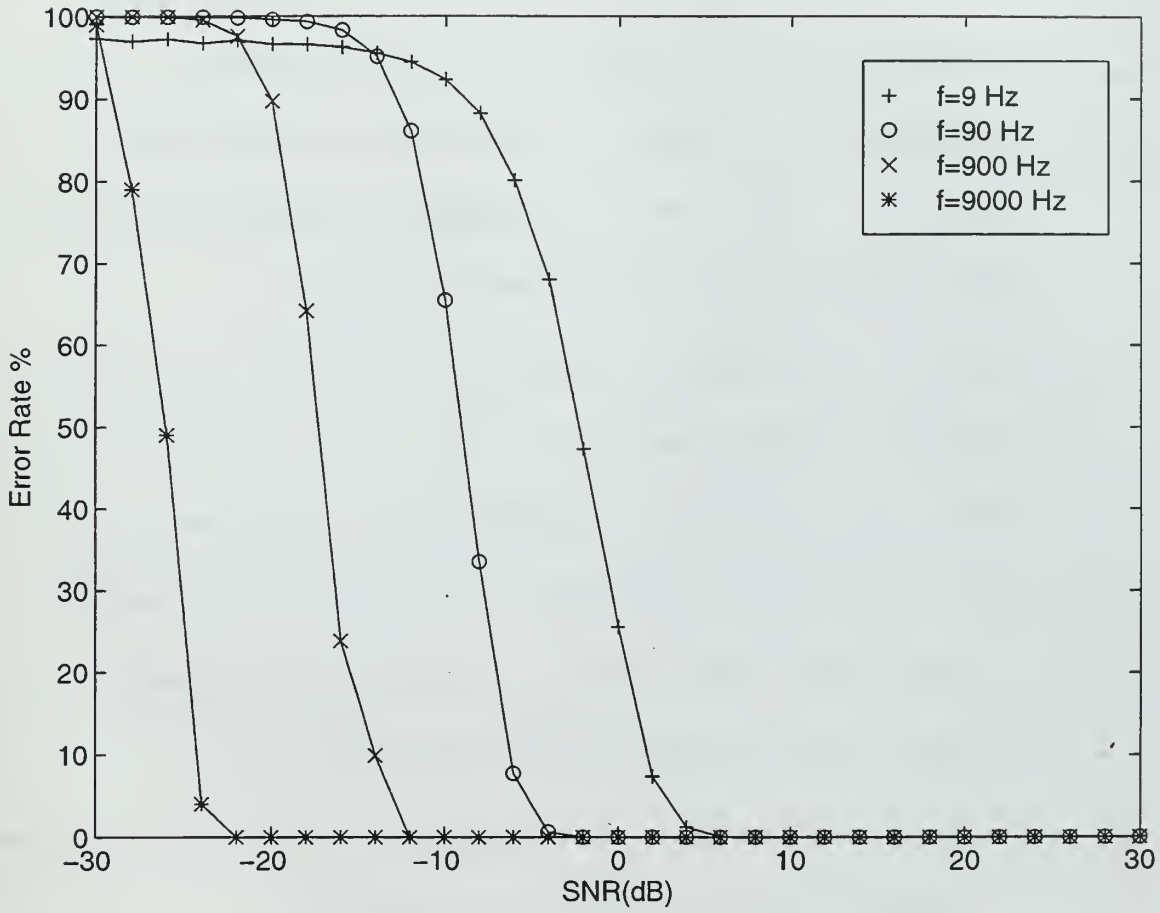


Figure 10: Error Rates vs. SNR for two-channel system

The following observations are made:

- As expected, the error rates improve as the SNR increases. A tradeoff between SNR and error rate is required.
- Improvements in error rates were obtained when higher frequencies were used. This is because at higher frequencies, higher sampling frequencies are required. This leads to a higher N-point DFT (higher gain) which is less affected by noise.
- However, at higher frequencies, the time taken to compute the DFT was much longer. To reduce the time taken, the following methods can be implemented:
  - If N is highly composite (factorable into powers of many small prime factors, preferably primes < 10), use a "mixed-radix" FFT implementation.
  - If N is prime, or contains very large prime factors, use the "chirp-z" transform.
  - Use three or more channels in the receiver. A three-channel receiver has a higher dynamic range for the same magnitude of sampling frequencies. For example, a two-channel receiver with sampling

frequencies 6 and 7 has a dynamic range of [0:9] while a three-channel receiver with sampling frequencies of 5, 6 and 7 has a dynamic range of [0:21].

E. THREE-CHANNEL ALGORITHM

The three-channel algorithm is similar to the two-channel algorithm as shown in Figure 11. The MATLAB code can be found in Appendix A.

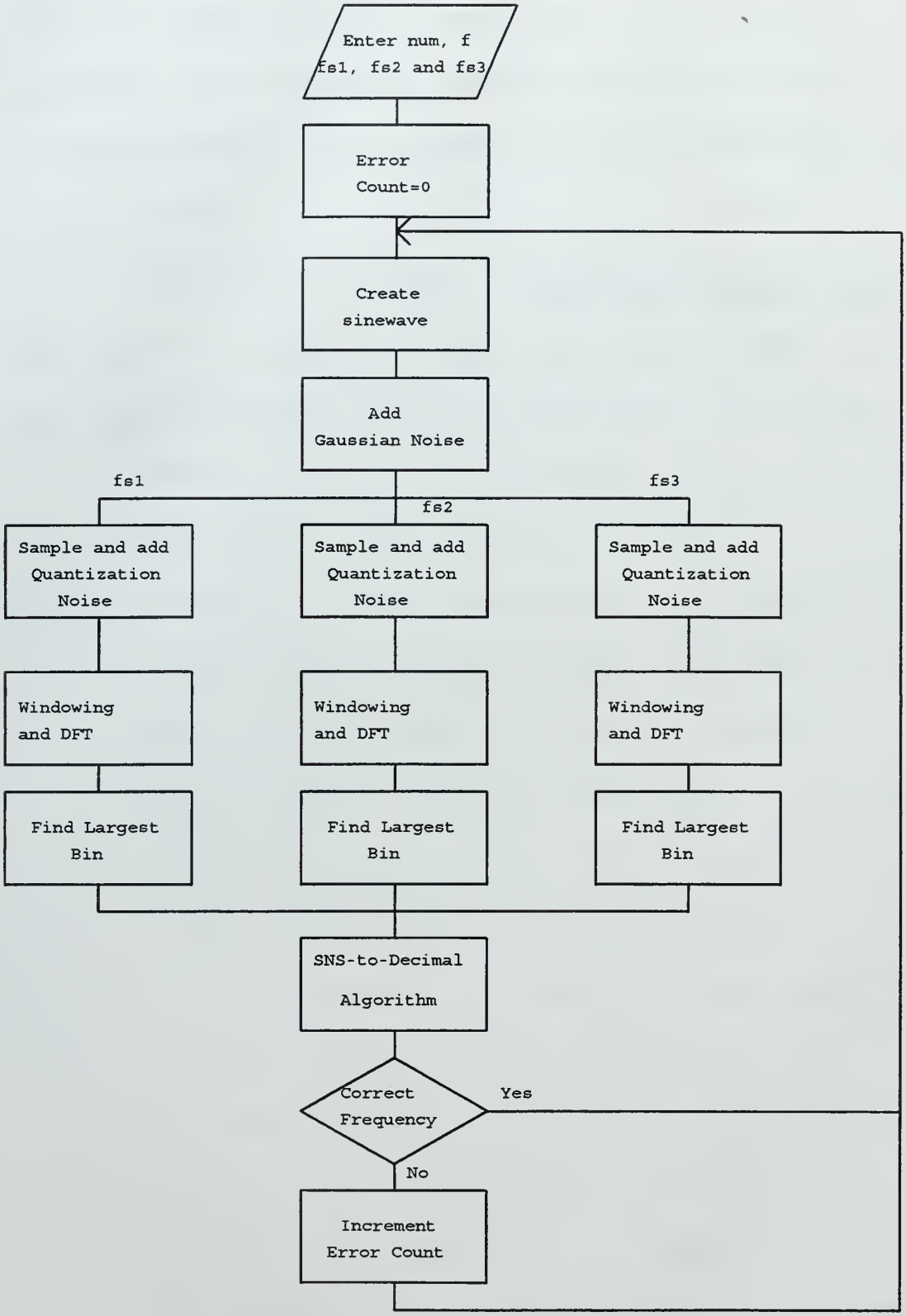


Figure 11: Three Channel Algorithm

F. TESTING OF THREE CHANNEL ALGORITHM

To test the three-channel case, the program is run with some of the following input and sampling frequencies in Table 4.

f	f_{s1}	f_{s2}	f_{s3}	Dynamic Range	Remarks
13	5	6	7	0:21	Low input frequency
100	17	18	19	0:171	Consecutive sampling frequencies
1040	17	91	919	0:1232	Sampling frequencies far apart
12125	90	929	937	0:42741	High input frequency

Table 4: Tested Input and Sampling Frequencies.

Apart from the anomaly discussed in the two-channel case, the algorithm works well in this noise-free (high signal-to-noise ratio) environment.

G. SIMULATION PARAMETERS FOR THREE-CHANNEL CASE

To obtain the error rates in a noisy environment, the three-channel software is run with the following parameters:

- Number of iterations, num = 10000

- Signal to Noise Ratio, SNRDB = -30 to 30 dB
- ADC resolution, bit = 14
- Input and sampling frequencies as shown in Table 5.

f	f_{s1}	f_{s2}	f_{s3}
9	5	7	11
90	13	14	17
900	41	42	43
9000	141	142	143

Table 5: Input and Sampling frequencies.

H. RESULTS FOR THREE-CHANNEL CASE

The results obtained are shown in Figure 12.

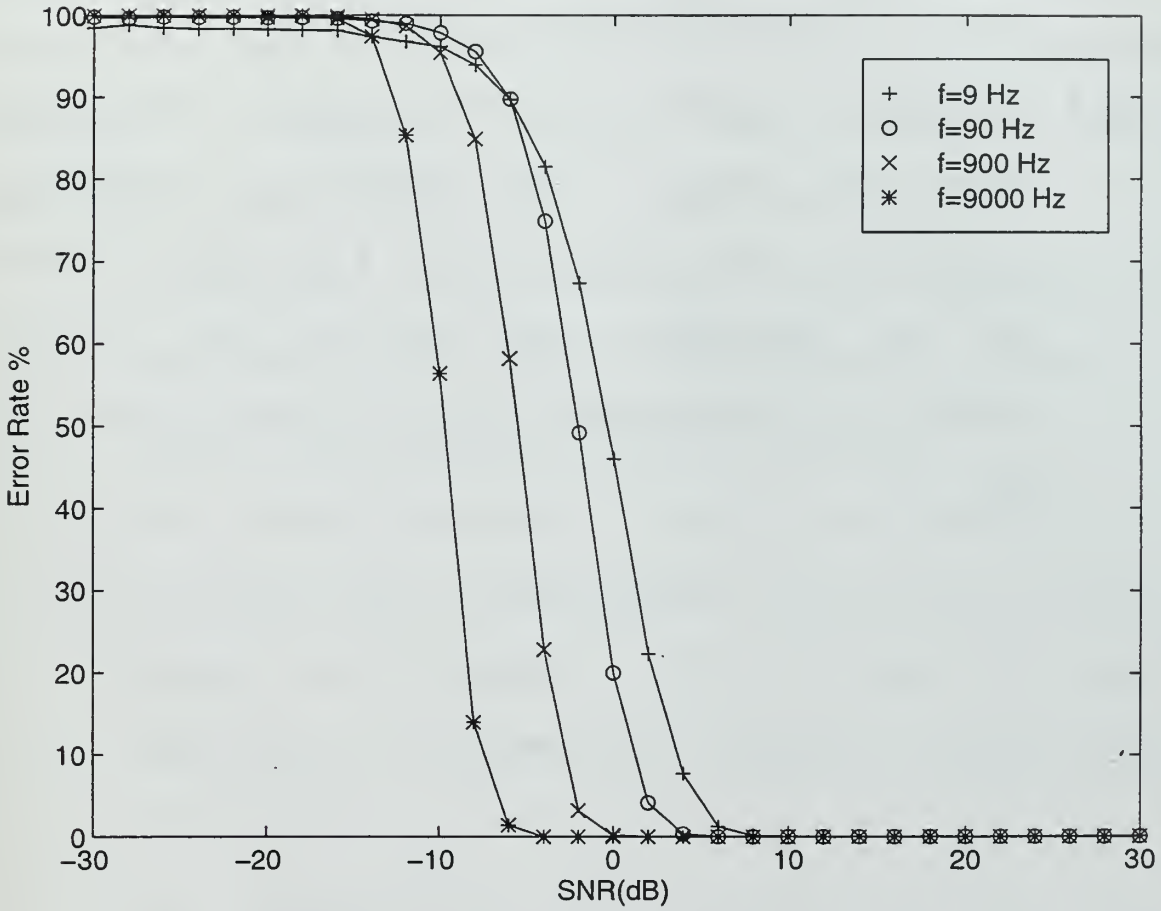


Figure 12: Error Rates vs. SNR for three-channel system

Comparing the two-channel and three-channel cases, the following observations can be made:

- The three-channel system is much faster than the two-channel system since the DFTs required are smaller due to the smaller sampling frequencies.
- However the results for the two-channel system with noise are better. For example to achieve a relatively error-free system for a frequency of 9000 Hz, the two-channel case requires only -22 dB. However, the three-channel case requires at least -4 dB.

IV. HARDWARE DESIGN AND FINDINGS

A. INTRODUCTION

In the last chapter, the instantaneous measurement of frequency using the SNS-to-decimal algorithm was verified. There is a need to investigate the implementation of the algorithm in hardware. Digital Signal Processing (DSP) hardware was selected for the following reasons:

- A major part of the algorithm is the processing of DFTs which is a digital signal processing task well suited to be carried out by DSP hardware.
- DSP hardware provides a fast way to implement the algorithm. The DSP development kit is easy to learn, program and simulate. It is ideal for this application to investigate hardware problems and limitations.
- Cost consideration: the development kit plus tools cost $1500;
- EW receivers are likely to incorporate DSP hardware.

B. TI TMS320C54X DSP DEVELOPMENT KIT

The TMS32054C54x DSKplus [Ref. 8-12] is a low cost DSP starter kit that gives a designer a working knowledge of DSP code to build DSP based systems. The development kit contains a stand-alone application board that can be connected to the PC. It executes code in real time at 40 MIPS while the Windows-based debugger analyzes it line-by-line, displaying internal DSP register information in multiple windows and in real time. It has an Analog Interface Circuit for the input of signals. The board's communication interface enables the creation of C54x DSP code and host PC code. Moreover, the hardware enables the use of expansion slots for adding memory, peripherals such as interface logic, other DSPs etc. The developed code can eventually be loaded into a resident DSP processor, which may be part of a EW receiver architecture. Figure 13 shows a block diagram of the development kit. A more detailed description of the kit can be found in Appendix B.

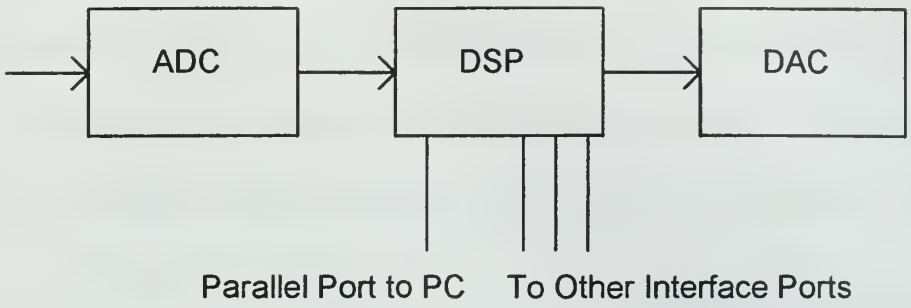


Figure 13: Block Diagram of DSP Hardware.

C. SOFTWARE

The software for the two-channel case described in Chapter II (Figure 2) is written using the DSP development kit. The software (found in Appendix C) is coded in 'C' language/assembly language and converted to the C54x assembly language (if required) prior to execution:

- Firstapp1.c/Firstapp2.c. These two programs poll the input channel and sample the input signal at the two sampling frequencies respectively.
- Hostapp1.cpp/Hostapp2.cpp. These two programs display the samples of the signals based on the two sampling frequencies and save the data in text files.

- Main.c. This program reads the data, executes the DFT, obtains the largest values for the two channels and then carries out a SNS-to-decimal conversion.

These programs were run individually and consecutively.

D. TESTING AND RESULTS

Using data generated by MATLAB, the main program was tested successfully in the development kit. The programs were then run with an input frequency of 126 Hz and sampling frequencies, 125 Hz and 128 Hz. Results obtained were intermittent i.e., correct results were not always obtained. A frequency counter and an oscilloscope were set up and it was found that the sampling frequencies were not stable. Testing with different frequencies did not improve the results.

E. PROBLEMS

Several problems were encountered during the investigation:

- Stability of Sampling Frequencies. The development kit carries out frequency division of the master oscillator to obtain the sampling frequencies. Unfortunately, the crystal oscillator has a

resolution of 5-10 Hz. This is unacceptable as a shift of 1 Hz in the sampling frequency will cause erroneous results. Moreover, the fact that the sampling frequencies are factors of the oscillator frequency and that they need to be pairwise relatively prime severely limits the choice of frequencies. A possible solution is to obtain the sampling frequencies directly from stable signal sources.

- DFT. For higher frequencies, the execution of the DFT takes a long time. Several solutions were suggested and discussed in the previous chapter.
- Memories. Insufficient memory error messages were encountered when high frequencies were used. The same messages occurred when attempts were made to run the routines together. More memories and/or more efficient DFT algorithms are required.

V. CONCLUDING REMARKS

The main contribution of this thesis is the verification of the relationship of the DFT to the SNS to resolve undersampling ambiguities and the investigation of hardware implementation issues using a DSP platform. Error rates for different SNR are also obtained.

The use of undersampling technique using the SNS to measure frequency is a viable method to implement in a EW receiver architecture. However, the need for faster DFT computation and stable sampling frequencies must be taken into account before they can be considered for incorporation into EW receivers. There is also a trade-off between the number of channels and SNR. For faster response, a multi-channel case is recommended; but a higher SNR is required.

LIST OF REFERENCES

1. J.L. Brown Jr., " On the Uniform Sampling of a Sinusoidal Signal," *IEEE Trans. Aerospace and Electronic Systems*, Vol. 24, no. 1, pp. 103-106, Jan 1988.

2. C. M. Rader, " Recovery of Undersampled Periodic Waveforms," *IEEE Trans. Acoustic, Speech and Signal Proceedings*, Vol. ASSP-25, no. 3, pp. 242-249, Jun 1977.

3. P. E. Pace, R. Leino and D. Styer, " Use of the Symmetrical Number System in Resolving Single Frequency Undersampling Aliases," *IEEE Trans. on Signal Processing*, Vol. 45, No. 5, May 1997.

4. G. Hill, " The Benefits of Undersampling," *Electronic Design*, pp. 69-79, July 1994.

5. P. E. Pace, P. A. Ramamoorthy, and D. Styer, " A Preprocessing Architecture for Resolution Enhancement in High Speed Analog to Digital Converter," *IEEE Trans. Circuits and Systems II: Analog and Digital Signal Processing*, Vol. 41, pp. 373-379, Jun 1994.

6. J. G. Proakis and D. G. Manolakis, " *Digital Signal Processing: Principles, Algorithms and Applications*," Prentice Hall, New Jersey, 1996.

7. A. M. Kirch, " *Elementary Number Theory - A Computer Approach*," Intext Educational Publishers, New York, 1974.

8. Texas Instruments, " *TMS320C54x DSKplus: DSP Starter Kit*," Literature Number: SPRU191, 1996.

9. Texas Instruments, " *TMS32054x DSP Reference Set*" , Literature Number: SPRU210, 1996.

10. Texas Instruments, " *TMS32054x Fixed Point Digital Signal Processor*," Literature Number: SLAS057, 1996.

11. Texas Instruments, " *TMS32054x Optimizing C Compiler User's Guide*," Literature Number: SPRU103, 1995.

12. Texas Instruments, " *TMS32054x Assembly Language Tools User's Guide*," Literature Number: SPRU102, 1996.

APPENDIX A

MATLAB CODE FOR SOFTWARE ALGORITHM

```
%
% Thesis Project
%
% Two Channel Receiver
%
% Note: The sampling frequencies should be relatively prime
%

clear all;

% Initialization

num=input('Enter Number of iterations:');       % Number of iterations
f=input('Enter Input Frequency:');              % Frequency of signal
fs1=input('Enter Sampling Frequency 1:');       % Sampling frequency 1
fs2=input('Enter Sampling Frequency 2:');       % Sampling frequency 2
fp1=fopen('c:\matlab\bin\thesis\result.dat','at'); % Store results

% Quantization levels

bit=14;
qnlevel=2^bit-1;                          % No. of quantization levels
q=2/qnlevel;                              % quantization size

for SNRDB=-30:2:30                        % Set Signal to Noise Ratio
                                          % from -30 dB to 30 dB

count=0;                                  % Error Count

for i=1:num
      SNR=10^(SNRDB/10);                  % Convert to non-dB units
      sigmasq=1/2/SNR;                    % Noise normalization assuming
                                          % signal power of 1
      t=(0:.001:1);
      sig=sin(2*pi*f*t);                  % signal
      t1=1/fs1:1/fs1:1;                   % first ADC
      noise1=sqrt(sigmasq)*randn(1,length(t1)); % noise
      ADCsig1=sin(2*pi*f*t1)+noise1;      % digitized signal
      ADCsig1=fix(ADCsig1/q)*q;           % quantized signal
      t2=1/fs2:1/fs2:1;                   % second ADC
      noise2=sqrt(sigmasq)*randn(1,length(t2)); % noise
      ADCsig2=sin(2*pi*f*t2)+noise2;      % digitized signal
      ADCsig2=fix(ADCsig2/q)*q;           % quantized signal

      %figure(1)
      %subplot(3,1,1), plot(t,sig)
```

```
%title('Figure 1. Plot of signal')
%xlabel('Time')
%ylabel('Amplitude')
%subplot(3,1,2),plot(t1,ADCsig1)
%title('Figure 2. Plot of sampled signal (sampling frequency 1)
plus noise')
%xlabel('time')
%ylabel('magnitude')

%subplot(3,1,3),plot(t2,ADCsig2)
%title('Figure 3. Plot of sampled signal (sampling frequency 2)
plus noise')
%xlabel('time')
%ylabel('magnitude')

% Window operation
% Assume rectangular window

winsize1=fs1;                  % size of window is fs1
winsize2=fs2;                  % size of window is fs2
winsig1=ADCsig1(1:winsize1);   % windowed sampled signal 1
winsig2=ADCsig2(1:winsize2);   % windowed sampled signal 2

% DFT Operation

DFTsig1=abs(fft(winsig1,winsize1));
DFTsig2=abs(fft(winsig2,winsize2));
DFTsig1a=DFTsig1(1:length(DFTsig1)/2 +1); % Taking half of image
DFTsig2a=DFTsig2(1:length(DFTsig2)/2 +1); % Taking half of image

%figure(2)
% Plot to locate position of maximum value
% Note that due to MATLAB (which cannot have a zero index, the
actual location is one less
%subplot(2,1,1), stem(DFTsig1a)
%title('Figure 1. DFT plot of signal with sampling frequency 1')
%xlabel('frequency bins')
%ylabel('magnitude')
%subplot(2,1,2),stem(DFTsig2a)
%title('Figure 2. DFT plot of signal with sampling frequency 2')
%xlabel('frequency bins')
%ylabel('magnitude')

% bin detector

[i,y1]=max(DFTsig1a);          % y1, y2 are locations of max values
[j,y2]=max(DFTsig2a);          % Note that due to MATLAB, the
                               % actual location is one less.
a1=y1-1;
a2=y2-1;
% SNS to Decimal Algorithm
```

```
      % To solve for f==ai(mod mi) (where "==" indicates congruence and
      % mi are pairwise relatively prime), the Chinese Remainder Theorem
      % states that there is a unique solution modulo M=m1*m2....mr.

      % A standard method of solution is to find integers bi such that
      % M*bi/mi==1(mod mi) where i=1,2,....r in which the solution is
      % f==M*b1*a1/m1 + M*b2*a2/m2 + ... + M*br*ar/mr (mod M)

      % For a 2 channel case, i.e. i=1,2,
      %           m2*b1 == 1(mod m1)
      %           m1*b2 == 1(mod m2)
      %               f == a1(mod m1)
      %               f == a2(mod m2)
      %               f == m2*b1*a1 + m1*b2*a2 (mod m1*m2)

      % Given m1(sampling frequency 1) and m2 (sampling frequency 2), to
      % find b1 and b2, the congruence equation is transformed to a
      % diphantine equation and solved using the Euclidean algorithm:
      %           m2*b1 - m1*y1 = 1
      %           m1*b2 - m2*y2 = 1
      % The above two equations can be combined into
      %           m2*b1 - m1*b2 = 1
      % b1 and b2 are solved by the function "lde.m" which is called by %
      "glde.m".
      %
      % f == a1(mod m1) and f == a2(mod m2) is solvable only if the
      % greatest common divisor of m1 and m2 divides (a2 - a1).
      % To solve for f, r from the diophantine equation
      % r*m1+s*m2 = a2 - a1 must be solved.
      % r is obtained from "glde.m" and f is calculated by the
      % equation f = a1+r*m1

      idiff=a2-a1;
      r = glde(fs1,fs2,idiff);
      freq=abs(a1+r*fs1);

      % Count the number of correct results.

      if freq==f
            count=count+1;
      end
end
error = 1-count/num;

% Write results to file

x1=fprintf(fp1,'%d %d %d %d %d %d\n', f, fs1, fs2, SNRDB, num, error);

plot(SNRDB,error,'y+')
title('Error Rate vs. Signal to Noise Ratio')
```

```
xlabel('SNR(dB)')
ylabel('Error Rate %')
hold on
end
fclose(fp1);

% To calculate the dynamic range

if rem(fs1,2)==0                    % To check whether fs1 is even
      DR=fs1/2 + fs2;
elseif rem(fs2,2)==0                % To check whether fs2 is even
      DR=fs2/2 + fs1;
else
      DR=.5*(fs1+fs2);              % fs1 and fs2 are odd numbers
end
%
% Thesis Project
%
% Three Channel Receiver
%
%

clear all;
close

% Initialization

num=input('Enter Number of iterations:');       % Number of iterations

f=input('Enter Input Frequency:');              % Frequency of signal

fs1=input('Enter Sampling Frequency 1:');       % Sampling frequency 1

fs2=input('Enter Sampling Frequency 2:');       % Sampling frequency 2

fs3=input('Enter Sampling Frequency 3:');       % Sampling frequency 3

fp1=fopen('c:\matlab\bin\thesis\result.dat','at');
% Store results in file for later processing if required

% Quantization levels

% bit=input('Enter ADC resolution:');
bit=14;
qnlevel=2^bit-1;                    % No. of quantization levels
q=2/qnlevel;                        % quantization size

for SNRDB=-30:2:30                  % Set Signal to Noise Ratio from -30
                                    % dB to 30 dB

count=0;
```

```
for i=1:num
      SNR=10^(SNRDB/10);              % Convert to non-dB units
      sigmasq=1/2/SNR;                % Noise normalization assuming
                                      % signal power of 1

      t1=1/fs1:1/fs1:1;                           % first ADC
      noise1=sqrt(sigmasq)*randn(1,length(t1)); % noise
      ADCsig1=1000*(sin(2*pi*f*t1)+noise1);     % digitized signal
      ADCsig1=fix(ADCsig1/q)*q;                 % quantized signal

      t2=1/fs2:1/fs2:1;                           % second ADC
      noise2=sqrt(sigmasq)*randn(1,length(t2)); % noise
      ADCsig2=1000*(sin(2*pi*f*t2)+noise2);     % digitized signal
      ADCsig2=fix(ADCsig2/q)*q;                 % quantized signal

      t3=1/fs3:1/fs3:1;                           % third ADC
      noise3=sqrt(sigmasq)*randn(1,length(t3)); % noise
      ADCsig3=1000*(sin(2*pi*f*t3)+noise3);     % digitized signal
      ADCsig3=fix(ADCsig3/q)*q;

      %figure(1)

      %subplot(3,1,1),plot(t1,ADCsig1(1:fs1))
            %title('Figure 1. Plot of sampled signal (sampling frequency
      1) plus noise')
      %xlabel('time')
      %ylabel('magnitude')
      %subplot(3,1,2),plot(t2,ADCsig2(1:fs2))
      %title('Figure 2. Plot of sampled signal (sampling frequency 2)
      plus noise')
      %xlabel('time')
      %ylabel('magnitude')
      %subplot(3,1,3),plot(t3,ADCsig3(1:fs3))
      %title('Figure 3. Plot of sampled signal (sampling frequency 3)
      plus noise')
      %xlabel('time')
      %ylabel('magnitude')

      % Window operation
      % Assume rectangular window

      winsize1=fs1;                       % size of window is fs1
      winsize2=fs2;                       % size of window is fs2
      winsize3=fs3;                       % size of window is fs3
      winsig1=ADCsig1(1:winsize1);        % windowed sampled signal 1
      winsig2=ADCsig2(1:winsize2);        % windowed sampled signal 2
      winsig3=ADCsig3(1:winsize3);        % windowed sampled signal 3

      % DFT Operation
      DFTsig1=abs(fft(winsig1,winsize1));
```

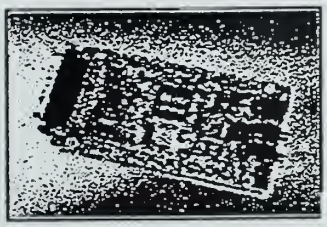
```
DFTsig2=abs(fft(winsig2,winsize2));
DFTsig3=abs(fft(winsig3,winsize3));

DFTsig1a=DFTsig1(1:length(DFTsig1)/2 +1); % Taking half the image
DFTsig2a=DFTsig2(1:length(DFTsig2)/2 +1); % Taking half the image
DFTsig3a=DFTsig3(1:length(DFTsig3)/2 +1); % Taking half the image

%figure(2)
%Plot to locate position of maximum value
%Note that due to MATLAB (which cannot have a zero index, the
%actual location is one less

%subplot(3,1,1), stem(DFTsig1a)
%title('Figure 1. DFT plot of signal with sampling frequency 1')
%xlabel('frequency bins')
%ylabel('magnitude')

%subplot(3,1,2),stem(DFTsig2a)
%title('Figure 2. DFT plot of signal with sampling frequency 2')
%xlabel('frequency bins')
%ylabel('magnitude')

%subplot(3,1,3),stem(DFTsig3a)
%title('Figure 3. DFT plot of signal with sampling frequency 3')
%xlabel('frequency bins')
%ylabel('magnitude')

% bin detector

[i,y1]=max(DFTsig1a);    % y1, y2 and y3 are the locations of
                         % maximum values
[j,y2]=max(DFTsig2a);    % Note that due to MATLAB, the actual
                         % location is one less.
[k,y3]=max(DFTsig3a);

a1=y1-1;
a2=y2-1;
a3=y3-1;

% SNS to Decimal Algorithm

b1=lde(fs2*fs3,fs1);
b2=lde(fs1*fs3,fs2);
b3=lde(fs1*fs2,fs3);

c1=b1*fs2*fs3;
c2=b2*fs1*fs3;
c3=b3*fs1*fs2;

freqmat=[a1*c1+a2*c2+a3*c3;a1*c1+a2*c2-a3*c3; a1*c1-a2*c2+a3*c3;
        a1*c1-a2*c2-a3*c3;-a1*c1+a2*c2+a3*c3;-a1*c1+a2*c2-a3*c3;
```

```
            -a1*c1-a2*c2+a3*c3;-a1*c1-a2*c2-a3*c3];

      freqmat=rem(freqmat,fs1*fs2*fs3);

      for i=1:8
            if (freqmat(i)<0)
               freqmat(i)=freqmat(i)+fs1*fs2*fs3;
            end
      end
      freq=min(abs(freqmat));

      % Count the number of correct results.

      if freq==f
            count=count+1;
      end
end
error = 1-count/num;

% Write results to file

x1=fprintf(fp1,'%d %d %d %d %d %d %d\n', f, fs1, fs2, fs3, SNRDB, num,
error);

plot(SNRDB,error,'y+')
hold on
end

fclose(fp1);

% To calculate the dynamic range

if rem(fs1,2)==0                    % To check whether fs1 is even
      x=[fs1/2 + fs2*fs3; fs1*fs2/2 + fs3; fs1*fs3 + fs2];
elseif rem(fs2,2)==0                % To check whether fs2 is even
      x=[fs2/2 + fs1*fs3; fs1*fs2/2 + fs3; fs2*fs3 + fs1];
elseif rem(fs3,2)==0                % To check whether fs3 is even
      x=[fs3/2 + fs2*fs1; fs3*fs2/2 + fs1; fs1*fs3 + fs2];
else                                % fs1,fs2 and fs3 are odd
      x=1/2*[fs1 + fs2*fs3; fs2 + fs1*fs3; fs3 + fs2*fs1];
end
DR=min(x);
```

```
% This function solves the general linear diophantine equation
% m2*b1 - m1*b2 = k and returns the value b1

function a=glde(m1,m2,k)

% Calls function "lde" to calculate b1, b2 and na

[b1,b2,na]=lde(m1,m2);

% To check whether the equation is solvable.
% na must be a factor of k for the equation to be solvable.

mult=k/na;

if (k-mult*na)==0 % Equation is solvable

      b1=b1*mult; % These new values solve the diophantine equation
      b2=b2*mult;

      mtest=b1;   % To check whether b1 and b2 are the least values
      md1=m1/na;  % that satisfies the diophantine equation
      md2=m2/na;
      mx=b1;
      mx=mx+md2;

      while (abs(mx)-abs(b1))<0
            b1=mx;
            b2=b2-md1;
            mx=mx+md2;
      end

      if (mtest-b1)==0
            mx=b1;
            mx=mx-md2;
            while (abs(mx)-abs(b1))<0
                  b1=mx;
                  b2=b2+md1;
                  mx=mx-md2;
            end
      end
end

a=b1;
```

```
% This function solves the linear diophantine equation
% m1*b1 + m2*b2 = na where m1 and m2 are the sampling frequencies
% and na is the greatest common divisor
% and returns the value b1, b2 and na
%
% m1 and m2 are assumed positive

function [b1,b2,na]=lde(m1,m2)

% Initialize bo1, bo2, b1 and b2

bo1=1;
bo2=0;
b1=0;
b2=1;

% Place m1 and m2 in ma(dividend) and na (divisor) respectively

ma=m1;
na=m2;

% Calculate quotient and remainder

iquot=fix(ma/na);
irem=ma-na*iquot;

% If remainder is not zero, reset dividend and divisor

while irem>0
      bo3=bo1-iquot*b1;       % calculate new coefficients of m1 and m2
      bo4=bo2-iquot*b2;
      bo1=b1;                 % redefine bo1, bo2, b1 and b2
      bo2=b2;
      b1=bo3;
      b2=bo4;
      ma=na;                  % redefine dividend and divisor
      na=irem;
      iquot=fix(ma/na); % reapply Euclidean algorithm
      irem=ma-na*iquot;
end
```

APPENDIX B



TMS320C54x DSKplus

The 'C54x DSKplus is a low-cost design tool that gives designers a working knowledge of DSP code. From this foundation, designers can begin building complete 'C54x DSP-based systems. Priced at US $149, the 'C54x DSKplus (part no. TMDS32000L0) is available from TI authorized distributors.

The 'C54x DSKplus builds on TI's industry-leading line of low cost, easy-to-use DSP Starter Kit (DSK) development boards. The high-performance board features the TMS320C542 16-bit fixed-point DSP. Capable of performing 40 million instructions per second (MIPS), the 'C542 makes the 'C54x DSKplus the most powerful DSK development board on the market.

Other TMS320 DSKs include the 'C2x DSK, the 'C5x DSK, and the floating-point 'C3x DSK.

Key Features

The 'C54x DSKplus includes:

- 40 MIPS TMS320C542-based board
- TLC320AC01 Analog Interface Circuit (AIC)
- 'C54x DSKplus assembler, loader, Code Explorer debugger, and sample programs (3.5" disks)
- TMS320C54x CPU and Peripherals Reference Guide
- TMS320C54x Algebraic Assembler Instruction Set
- TMS320C54x Datasheet
- TMS320C54x DSKplus User's Guide
- TLC320AC01 Datasheet
- PC connector cable and universal power supply included
- US $149 discount coupon toward the purchase of the 'C54x EVM

DSKplus Key Features	Benefits
TMS320C542 DSP (40 MIPS, 16-bit)	High-performance, very efficient architecture requires fewer MIPS than competing DSPs to implement most algorithms.
Code Explorer debugger interface	An easy-to-use, true Windows-based interface. Supports symbolic debugging, breakpoints, graphical animation, variable watch windows, file I/O, algebraic/mnemonic disassembly, on-line help.
Symbolic debugging (Code Explorer)	Enables easy programmability by using labels for referencing constants, variables, matrices by name.
Algebraic assembler	Bypasses learning new DSP mnemonic instruction set specifics. Makes coding easier and more straight-forward. Easy one-step assembly and linking process.
Demo programs / Application code	Helps users get up-to-speed quickly
TLC320AC01 Analog Interface Chip	Low power dissipation, 14-bit linear resolution, programmable sampling rates, anti-aliasing filter, and input gain; selectable auxiliary input; data read-back
Socketed Programmable Array Logic (PAL)	Allows experienced designers to reprogram the PAL and change the way the host port interface works on the C54x DSKplus.
Universal power supply & cable included	Allows for immediate use out of the box; ideal for powering daughter cards; filtered and regulated - thus no need for on-board voltage regulation.

'C54x Algebraic Assembler

The C54x DSKplus includes the algebraic assembler that speeds the initial code development process. The algebraic assembler does not require new users to learn a new DSP mnemonic instruction set, making coding easier and more direct. The assembler also utilizes a one-step assembly and linking process to simplify code debugging. The software accomplishes this by using special directives to assemble code at an absolute address.

Some extremely useful features include:

- In-line Assembly expression analysis allows the assembler to work when defining complex variables or bit locations.
- Symbolic Debugging allows the user to reference variables by name instead of the physical address.
- Assembling conditional blocks of assembly code using .if/.else if/.end if directives. This is especially helpful when you want to conditionally assemble code via a command-line argument of internal assembly variable.
- Support of .sect, .bss, .usect, .text, and .data sections.

Code Explorer Debugger

The 'C54x DSKplus debugger was developed by GO DSP Corporation in an effort to provide the first true Windows-based debugger for a DSK. The Code Explorer debugger supports debugging, a new feature available only on the DSKplus that allows the user to specify labels for referencing constants, variables, and marticies by name. Also, the debugger desktop environment is fully configurable and loaded upon entry into the debugger. This means that optional colors, fonts, and window sizes can be changed within the debugger and saved upon exiting.

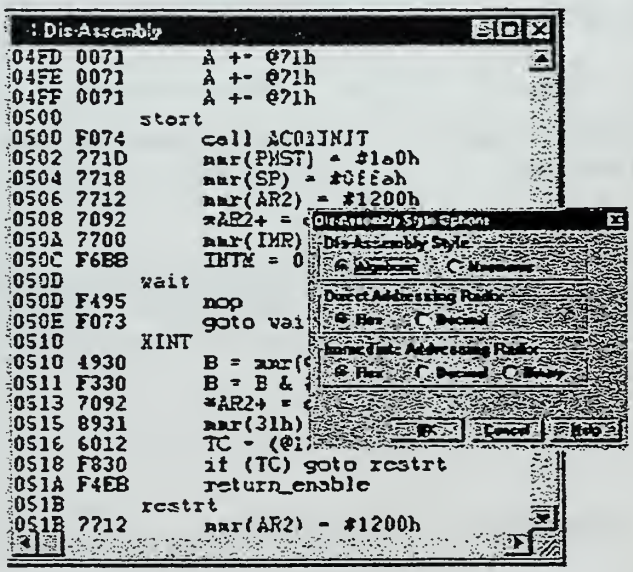
Some additional features of the debugger include capability of connecting files as I/O, graphical animation, and data memory viewing. The file I/O capability enables users to connect files as inputs or outputs to any location within your application code. Therefore you can simulate different input sequences and data streams without having to physically generate them.

Graphical animation allows you to view data in a graphical format, either with time domain or frequency domain and in a variety of variable sizes (i.e. 8-bit signed char, 8-bit unsigned char, 16-bit, 32-bit, etc).

Disassembly Window

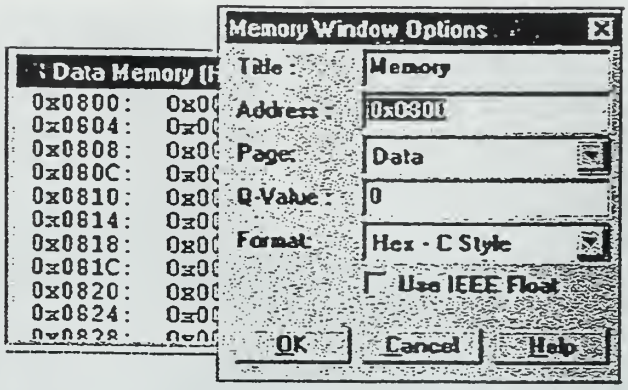
The disassembly window displays the DSP code in algebraic instructions. The variable names and subroutines (symbols) are shown in blue. The physical DSP address is the first column and the machine code for the instructions are in column 2. The yellow bar indicates the location where the DSP program counter (PC) points.

The disassembly window properties can be accessed by placing the cursor in the disassembly window and right-clicking and then choosing properties. The disassembly window can display code in algebraic or mnemonic formats with direct and immediate addressing values shown in hex, decimal and even binary.



Data Memory Window

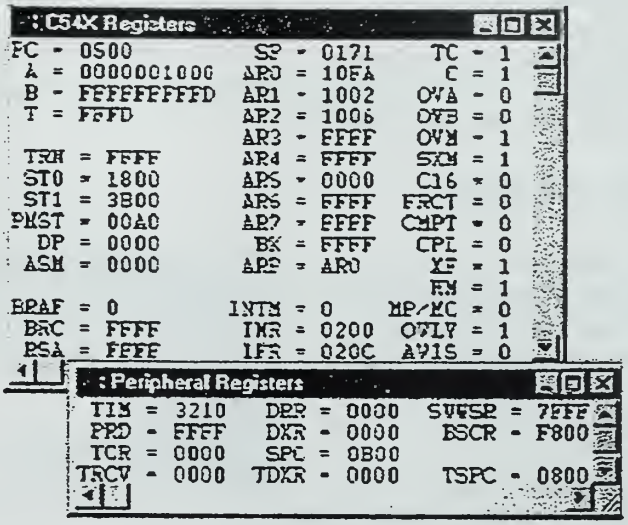
The data memory window can be modified or replicated as needed. By placing the cursor inside the data memory window and right-clicking and then choosing properties, the user can change the title of the window, starting address and even data organization in the window. Valid display formats include 8-bit signed/unsigned char, signed/unsigned long, floats, and others. The page field can specify either Data or Program memory spaces.



'C54x CPU and Peripheral Registers

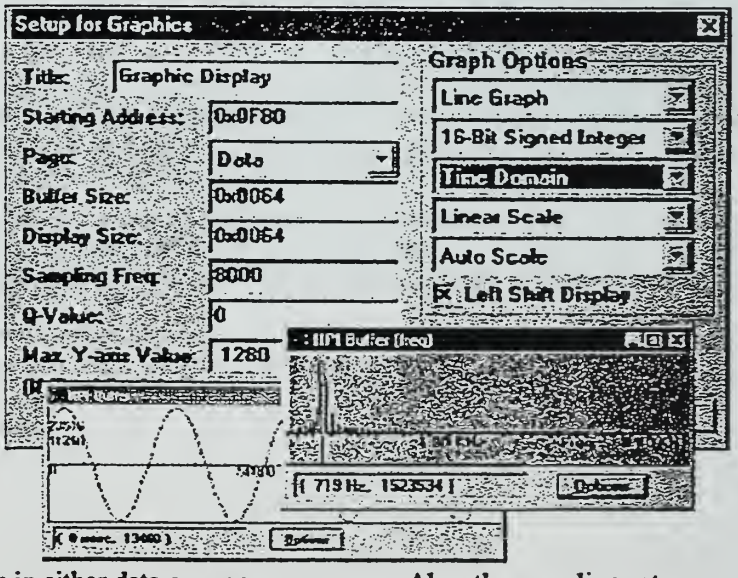
The two register windows in the 'C54x Code Explorer debugger are the CPU and Peripheral Registers. The 'C54x CPU Registers is the collection of registers which control the operation of the DSP CPU. The program counter, status register, and configuration registers are contained within this window. Notice that bit values within the register are brought out separately to make modification and monitoring easier.

The second window is the Peripherals window. This window includes the registers for configuring the DSP peripherals like the serial ports and timers. Modifications to this register can be done by clicking on the register in the Peripheral Registers window.



Graphical Windows

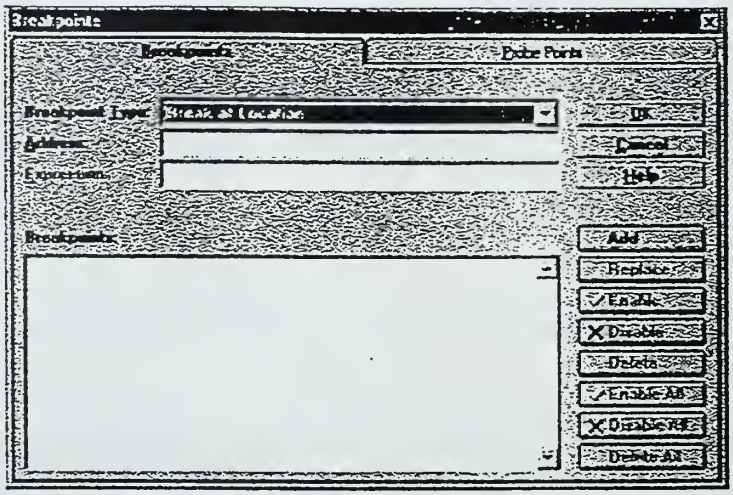
Graphical windows are extremely useful when trying to view a value of a register, variable, or buffer. The graphic window allows the user to animinate any value in either data or program DSP memory. This is accomplished by placing a breakpoint anywhere in the application code and pressing the Animation button. Each time the DSP reaches the breakpoint the graphical windows are updated and refreshed.



The options window contains the graphics setup for the window. For example, the title can be changed to reflect the data being animated, the display buffer length can be changed, or the data read from the DSP can either be a single value from a list (buffer) of values in either data or program memory. Also, the sampling rate can be modified for correct displaying of the frequency data (FFT). The display can be viewed using 8-bit signed/unsigned chars, ints, long, floats, and even a log can be performed on the displayed data.

Setting Breakpoints

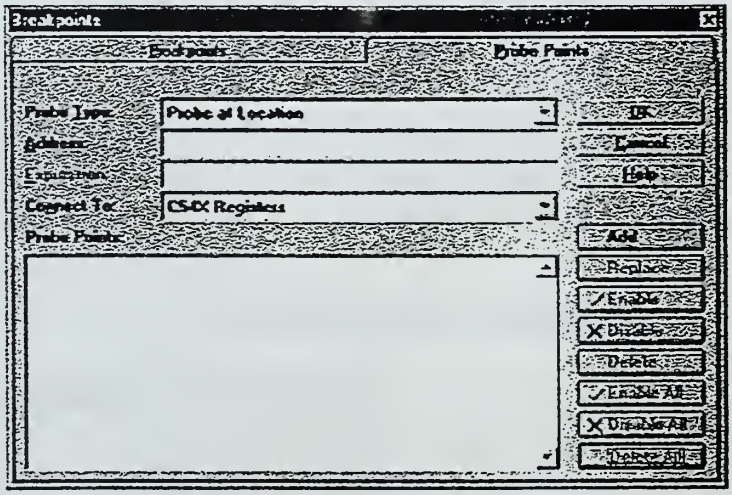
A breakpoint can be selected by either double clicking on a line in the Disassembly window or by Selecting the DEBUG-BREAKPOINTS in the Pull Down Menu. The Pull Down Menu will prompt you with a menu listing all the available symbols in the Symbols box. You can either select a breakpoint from the list of Symbols or by entering an address in the Address field.



The Breakpoint dialog box contains the following fields: Address, Symbols and Breakpoints. If the address of the desired breakpoint is known, simply enter the value in the Address field. The Symbol field contains the list of all the symbols in the program. If the location address of the breakpoint is labeled, simply type the label name and press add.

Setting Probe Points

Probe points allow the update of a particular window or the reading/writing of samples from a file to occur at a specific point in an algorithm. This effectively "connects a signal probe" to that point in the algorithm.

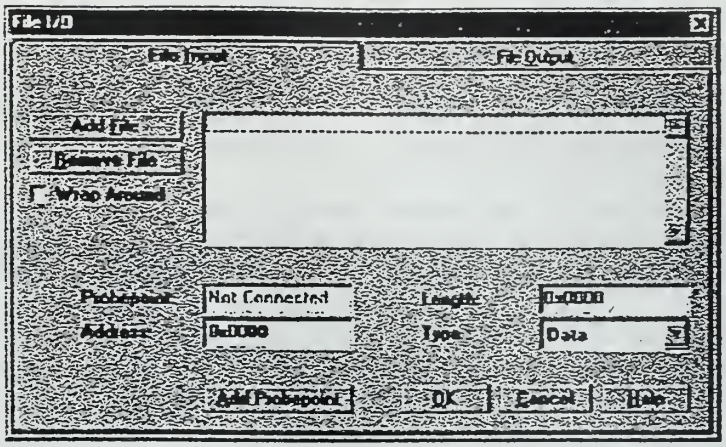


When a graph window object is created, it assumes that it is to be updated at every breakpoint. However, this attribute can be changed and the window can be updated only when the program reaches the connected probe point. After the probe point is hit, and the window is updated, execution of the program is continued. This optimizes the display of the graph window and also allows you to keep a history of the signal even when the data on the DSP is not valid.

With the combination of Code Explorer's File I/O capabilities, probe points can also used to connect streams of data to a particular point in the DSP Code. When the probe point is reached in the algorithm, data is streamed from a specific memory area to file, or from the file to memory.

Using File I/O

Code Explorer allows the user to stream data onto (or from) the target from a PC file. This allows the user to simulate code using known sample values. Note that this file I/O feature is not intended to satisfy real-time constraints. The File Input/Output feature uses probe points. When the execution of the program reaches a probe point, the connected object, whether it is a file, graph or memory window, is updated. Once the connected object is updated, execution continues. Using this concept, if a probe point is set at a specific point in the code and then connected to a file, file I/O functionalities can be implemented.

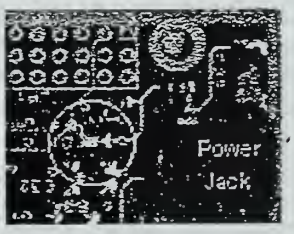


System Requirements

- A 386, 486, or Pentium PC with a 3.5" disk drive
- 4-bit parallel and/or 8-bit bidirectional parallel ports.
- A minimum of 4Mbytes of memory
- Color VGA monitor
- Windows 3.1 or Windows 95
- ASCII editor

How to Install

When connecting the DSKplus to your PC, it is highly recommended you turn off your PC's power to make the connections below:

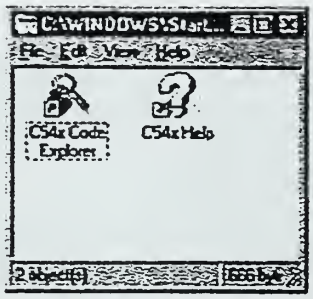


1. Connect the DB25 cable (female) to the PC's Parallel port (male).
2. Connect the DB25 cable (male) to the DSKplus board (female).
3. Connect the power cord (NEMA cable) to the 5 volt power supply.
4. Connect the 5-pin DIN-to-5.5mm adapter to the power supply's 5-pin DIN connector.
5. Plug the power supply power cord to the wall outlet.
6. Plug the 5.5mm connector into the power jack of the DSKplus board.

At this point the green power LED is illuminated and power is supplied to the 'C54x DSKplus board. If the Green LED is not illuminated, check the connections on the power supply and power cord.

Installing the software

The DSKplus kit includes two 3.5" floppies labeled Disk #1 and Disk #2. To install the software correctly, please follow the steps below:

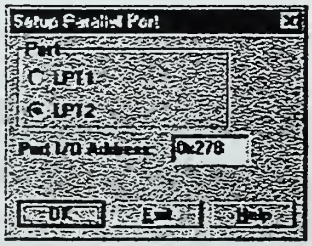


1. Insert Disk #1 into the 3.5" drive.
2. From the start menu (Windows95) or the Files menu (Windows 3.1) select the Run.. option. Type a:\setup.exe
3. The installation script will appear. You will be asked to select a destination directory. By default it will select the DSKplus directory. Enter the directory name if you would like to specify a different directory.
4. When prompted, insert Disk #2 into the 3.5" floppy drive.
5. When installation has completed, the installation will inform you that the installation was successful. At this point a Code Explorer Group will appear.

Starting the Debugger

To start the debugger, click on the icon located in the Code Explorer Group or desktop. The Code Explorer background and windows will appear with the Setup Box shown active.

Select the port which is connected to the DSKplus board. If for some reason the port is not listed, the port address can be modified by typing in the address int the text box.



As a result of selecting the correct port and proper hardware connections, the debugger will fill its windows with data and the DSKplus is now functioning. If for some reason the debugger responds with the error "Can't initialize Target DSP", follow the directions in the error box.

Troubleshooting

1. Is the power on? Be sure green LED is illuminated. If not, a loose power cable is hampering your setup.
2. Is the parallel port cable connection secure? In many new DSKplus boards and parallel port cables, substantial pressure many be needed to connect the cables. Connect the cable to the DSKplus board by placing the thumb behind the DB-25 connector. Take the cable connector chassis and place between the index and middle fingers. Align the connectors and press the fingers together.
3. The port selected is not being "Captured" by Windows 95. Capturing is used by Windows 95 to allow DOS programs access to printers. The port can be released by going into the control panel and selecting the printers icon. Highlight any printer and go to the File pulldown on the command bar. Select properties and then the Details tab. The Details tab includes a button named End Capture... Click on this button and select the LPT port where the DSKplus board is connected. If the LPT is not listed, then the port is not captured (select cancel) and proceed to number 4.
4. The port selected is configured as an EPP or ECP port. The DSKplus board supports 4-bit unidirectional and 8-bit bidirectional parallel ports. The DSKplus does not support EPP and ECP ports. To check the port configuration, exit out and reboot your system. At the point where the BIOS Setup routine can be selected, press the keyboard sequence to enter the BIOS (usually CTRL+ALT+ESC). Confirm that the parallel port is setup as '8-bit', 'bidirectional' or 'standard.' Specifically, not an EPP or ECP port. If problems persist, run the included selftest program.

Beyond the 'C54x DSKplus

With higher performance than any other DSK available today, the 'C54x DSKplus offers a rich development environment for benchmarking and evaluating code in real-time. The 'C54x DSKplus is designed as an easy-to-use entry into the world of high-performance fixed-point DSPs.

However, as your design experience grows, you may require additional functionality and expanded capabilities. To meet these needs, TI offers a comprehensive line of evelopment tools for the TMS320 DSPs that support the design process from system concept to production.

Other 'C54x Development Tools

APPENDIX C

C AND ASSEMBLY LANGUAGE CODE FOR DSP HARDWARE

```
;***************************************************************************
; File: FirstAp1.ASM
; When sampling frequency is changed, need to change
;       a. buffer size here
;       b. A and B registers in AC01ini1.asm
;       c. sampling frequency in dftsort.c
;       d. buffer size in hostapp1.cpp
;
;***************************************************************************
       .width   80
       .length  55
       .title "FirstApp program"
         .mmregs
       .setsect ".text",   0x500,0
       .setsect "vectors", 0x180,0
;=========================================================================
;
;  VECTORS
;
;=========================================================================
      .sect "vectors"
      .copy "c:\dskplus\inits\vectors.asm"
      .text
start:
      call AC01INIT
      pmst = #01a0h           ; set up iptr
      sp = #0ffah             ; init stack pointer.
      ar2 = #1200h            ; pointer to receive buffer at 1200h.
      *ar2+ = data(#0bh)      ; store to rcv buffer
      imr = #280h
      intm = 0                ; ready to rcv int's

wait  nop
      goto   wait

;      ---------------- Receive Interrupt Routine -----------------
XINT:
      b = trcv                ; load acc b with input
      b = #0FFFCh & b
      *ar2+ = data(#0bh)      ; store to rcv buffer
      tdxr = b                ; transmit the data.
        TC = (@ar2 == #1471h) ; change here if fs changes
        if (TC) goto restrt   ; stop if rcv buffer is at 1471h
      return_enable
restrt
```

```
        ar2 = #1200h            ; set intm bit  ...no int's
        hpic = #0ah             ; flag host task completed

      return_enable
;      ----------------------- end ISR ---------------------------

      .copy "c:\dskplus\firstapp\ac01ini1.asm"
      .end
;************************************************************************
; File: FirstApp.ASM -> First Application program for the 'C54x DSKplus
;       a. buffer size here
;       b. A and B registers in AC01ini2.asm
;       c. sampling frequency in dftsort.c
;       d. buffer size in hostapp2.cpp
;
;************************************************************************
       .width   80
       .length  55
       .title "FirstApp program"
       .mmregs
       .setsect ".text",   0x500,0
       .setsect "vectors", 0x180,0
;=======================================================================
;
;  VECTORS
;
;=======================================================================
      .sect "vectors"
      .copy "c:\dskplus\inits\vectors.asm"
      .text
start:
      call AC01INIT
      pmst = #01a0h             ; set up iptr
      sp = #0ffah               ; init stack pointer.
      ar2 = #1200h              ; pointer to receive buffer at 1200h.
      *ar2+ = data(#0bh)        ; store to rcv buffer
      imr = #280h
      intm = 0                  ; ready to rcv int's

wait  nop
      goto   wait

;      ---------------- Receive Interrupt Routine -----------------
XINT:
      b = trcv                          ; load acc b with input
      b = #0FFFCh & b
      *ar2+ = data(#0bh)                ; store to rcv buffer
      tdxr = b                          ; transmit the data.
      TC = (@ar2 == #01400h)            ; change here if fs change
      if (TC) goto restrt               ; stop if rcv buffer is at 1400h
      return_enable
```

```
restrt
      ar2 = #1200h                    ; set intm bit  ...no int's
      hpic = #0ah                     ; flag host task completed
      return_enable
;      ----------------------- end ISR ----------------------------

      .copy "c:\dskplus\firstapp\ac01ini2.asm"
      .end
```

```
;***************************************************************************
;
; File: AC01INI1.ASM -> AC01 Initialization Routine
;
;***************************************************************************
       .width   80
       .length  55
       .title "AC01 Initialization Program"
       .mmregs

*****************************************************************************
* Certain AC01 registers can be initialized using a conditional assembly
* constant. By setting the constant REGISTER to the appropriate value,
* the assembler will either include initialization for certain registers
* or ignore register initialization.
*
* The constant REGISTER should be set to include the following AC01
* register:
*
* REGISTER (binary) =
*
*     0000 0000 0000 0001  -> initialize Register 1  (A Register)
*     0000 0000 0000 0010  -> initialize Register 2  (B Register)
*     0000 0000 0000 0100  -> initialize Register 3  (A' Register)
*     0000 0000 0000 1000  -> initialize Register 4  (Amplifier Gain-
*                                                     Select)
*     0000 0000 0001 0000  -> initialize Register 5  (Analog
*                                                     Configuration)
*     0000 0000 0010 0000  -> initialize Register 6  (Digital
*                                                     Configuration)
*     0000 0000 0100 0000  -> initialize Register 7  (Frame-Sync Delay)
*     0000 0000 1000 0000  -> initialize Register 8  (Fram-Sync number)
*
* Any combination of registers can be initialized by adding the binary
* number to the REGISTER  constant. For example to initalize Registers 4
* and 5, REGISTER = 18h. Upon assembly, only code for register 4 & 5
* initialization is included in the AC01INIT module. When called the
* module will load REG4 and REG5 values into internal AC01 registers.
*
*
* Register 4 is always loaded to get a 6db input gain. This sets full-
* scale to 3v(p-p input) due to the single-ended AC01 configuration.
*
*
REGISTER .set   0bh     ; Powerup default values:
REG1     .set   1feh    ;*                 112h
REG2     .set   21fh    ;*                 212h
REG3     .set   300h    ;                  300h
REG4     .set   40dh    ;*                 405h
REG5     .set   501h    ;                  501h
REG6     .set   600h    ;                  600h
```

```
REG7     .set   700h    ;                    700h
REG8     .set   801h    ;                    801h

AC01INIT:
      xf = 0                    ; reset ac01
      intm = 1                  ; disable all int service routines
      tcr = #10h                ; stop timer
      imr = #280h               ; wakeup from idle when TDM Xmt int
      tspc = #0008h             ; stop TDM serial port
      tdxr = #0h                ; send 0 as first xmit word
      tspc = #00c8h             ; reset and start TDM serial port
      xf = 1                    ; release ac01 from reset

;     -------------- Register init's -----------------------------

      .eval REGISTER & 1h, SELECT   ; if REG1 then include this source
      .if SELECT = 1h               ;
      a = #REG1                     ; load Acc A with REG1 value
      call REQ2                     ; Call REQ2 subroutine
      .endif

      .eval REGISTER & 2h, SELECT   ; if REG2 then include this source
      .if SELECT = 2h
      a = #REG2
      call REQ2

      .endif

      .eval REGISTER & 4h, SELECT   ; if REG3 then include this source
      .if SELECT = 4h
      a = #REG3
      call REQ2
      .endif

      .eval REGISTER & 8h, SELECT   ; if REG4 then include this source
      .if SELECT = 8h
      a = #REG4
      call REQ2
      .endif

      .eval REGISTER & 10h, SELECT  ; if REG5 then include this source
      .if SELECT = 10h
      a = #REG5
      call REQ2
      .endif

      .eval REGISTER & 20h, SELECT  ; if REG6 then include this source
      .if SELECT = 20h
      a = #REG6
      call REQ2
      .endif
```

```
      .eval REGISTER & 40h, SELECT  ; if REG7 then include this source
      .if SELECT = 40h
      a = #REG7
      call REQ2
      .endif

      .eval REGISTER & 80h, SELECT  ; if REG8 then include this source
      .if SELECT = 80h
      a = #REG8
      call REQ2
      .endif
      return

REQ2
      ifr = #080h             ; clear flag from IFR
      tdxr = #03h             ; request secondary when AC01 starts

      idle(1)                 ; wait for primary to xmit
      tdxr = a                ; send register value to serial port
      ifr = #080h             ; clear flag from IFR

      idle(1)                 ; wait for secondary to xmit
      tdxr = #0h              ; send neutral state in case last init
      ifr = #080h             ; clear flag from IFR
      idle(1)                 ; wait for neutral state to xmit
      return                  ; return from subroutine
      .end
```

```
;***************************************************************************
;
; File: AC01INI2.ASM -> AC01 Initialization Routine
;
;***************************************************************************
       .width   80
       .length  55
       .title "AC01 Initialization Program"
       .mmregs

*****************************************************************************
* Certain AC01 registers can be initialized using a conditional assembly
* constant. By setting the constant REGISTER to the appropriate value,
* the assembler will either include initialization for certain registers
* or ignore register initialization.
*
* The constant REGISTER should be set to include the following AC01
* register:
*
* REGISTER (binary) =
*
*     0000 0000 0000 0001  -> initialize Register 1  (A Register)
*     0000 0000 0000 0010  -> initialize Register 2  (B Register)
*     0000 0000 0000 0100  -> initialize Register 3  (A' Register)
*     0000 0000 0000 1000  -> initialize Register 4  (Amplifier Gain-
*                                                     Select)
*     0000 0000 0001 0000  -> initialize Register 5  (Analog
*                                                     Configuration)
*     0000 0000 0010 0000  -> initialize Register 6  (Digital
*                                                     Configuration)
*     0000 0000 0100 0000  -> initialize Register 7  (Frame-Sync Delay)
*     0000 0000 1000 0000  -> initialize Register 8  (Fram-Sync number)
*
* Any combination of registers can be initialized by adding the binary
* number to the REGISTER  constant. For example to initalize Registers 4
* and 5, REGISTER = 18h. Upon assembly, only code for register 4 & 5
* initialization is included in the AC01INIT module. When called the
* module will load REG4 and REG5 values into internal AC01 registers.
*
*
* Register 4 is always loaded to get a 6db input gain. This sets full-
* scale to 3v(p-p input) due to the single-ended AC01 configuration.
*
*
REGISTER .set   0bh     ; Powerup default values:
REG1     .set   1feh    ;*                 112h
REG2     .set   23ch    ;*                 212h
REG3     .set   300h    ;                  300h
REG4     .set   40dh    ;*                 405h
REG5     .set   501h    ;                  501h
REG6     .set   600h    ;                  600h
```

```
REG7    .set   700h    ;                    700h
REG8    .set   801h    ;                    801h


AC01INIT:
      xf = 0                    ; reset ac01
      intm = 1                  ; disable all int service routines
      tcr = #10h                ; stop timer
      imr = #280h               ; wakeup from idle when TDM Xmt int
      tspc = #0008h             ; stop TDM serial port
      tdxr = #0h                ; send 0 as first xmit word
      tspc = #00c8h             ; reset and start TDM serial port
      xf = 1                    ; release ac01 from reset

;     --------------- Register init's -----------------------------

      .eval REGISTER & 1h, SELECT   ; if REG1 then include this source
      .if SELECT = 1h               ;
      a = #REG1                     ; load Acc A with REG1 value
      call REQ2                     ; Call REQ2 subroutine
      .endif

      .eval REGISTER & 2h, SELECT   ; if REG2 then include this source
      .if SELECT = 2h
      a = #REG2
      call REQ2

      .endif

      .eval REGISTER & 4h, SELECT   ; if REG3 then include this source
      .if SELECT = 4h
      a = #REG3
      call REQ2
      .endif

      .eval REGISTER & 8h, SELECT   ; if REG4 then include this source
      .if SELECT = 8h
      a = #REG4
      call REQ2
      .endif

      .eval REGISTER & 10h, SELECT  ; if REG5 then include this source
      .if SELECT = 10h
      a = #REG5
      call REQ2
      .endif

      .eval REGISTER & 20h, SELECT  ; if REG6 then include this source
      .if SELECT = 20h
      a = #REG6
```

```
      call REQ2
      .endif

      .eval REGISTER & 40h, SELECT  ; if REG7 then include this source
      .if SELECT = 40h
      a = #REG7
      call REQ2
      .endif

      .eval REGISTER & 80h, SELECT  ; if REG8 then include this source
      .if SELECT = 80h
      a = #REG8
      call REQ2
      .endif
      return

REQ2
      ifr = #080h             ; clear flag from IFR
      tdxr = #03h             ; request secondary when AC01 starts

      idle(1)                 ; wait for primary to xmit
      tdxr = a                ; send register value to serial port
      ifr = #080h             ; clear flag from IFR

      idle(1)                 ; wait for secondary to xmit
      tdxr = #0h              ; send neutral state in case last init
      ifr = #080h             ; clear flag from IFR
      idle(1)                 ; wait for neutral state to xmit
      return                  ; return from subroutine
      .end
```

```
/*********************************************************************/
/*
/* File: HOSTAPP1.CPP Source code for host application
/*
/*********************************************************************/
#include <HI54X.H>
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
extern int  datareg[],statreg[],ctrlreg[];
extern int  pport, portmode,Readdelay;

void main(void)
{
      FILE *fp;
      if ((fp=fopen("data1.dat","w"))==NULL)       /* Open file  */
      {
            clrscr();
            printf("Cannot open file .\n");
            exit(0);
      }
            portmode=0;                          /* 4-bit mode */
            Readdelay = 20;                      /* In case host slow*/
            clrscr();                            /* Clear the screen */
            if((pport=locate_port()) >= 5){  /* Find the port.   */
                     printf("No connection\n"); /* If no connection */
                     backout();                 /* then leave board */
                     exit(0);}                  /* in known state   */
            else{}
            _setcursortype(_NOCURSOR);           /* Hide text cursor */
            set_latch(1,1);                      /* Keep DSP running */
            int word =0, col=0;                  /* and bring PAL out*/
                                                 /* out of Tri-state */
                  col=0;
                  gotoxy(1,1);                   /* go to home       */
                  send_word(0x0808, C_SEND);     /* Clear the HINT   */
                  HINT(10000);                   /* Wait for nxt HINT*/
                  send_word(0x1200, A_SEND);     /* Goto 0x46 entries*/
                                                 /* before buffer    */
                  for(int buf=0 ; buf < 0x271; buf++)
                                          /* change here if fs change*/
                    {
                     word = read_word(D_READ);  /* Read word from pp*/
                     printf("%4.4x ", word);    /* Print it to scr  */
                     fprintf(fp, "%d\n", word); /* Output to file   */
                     if(col >= 13){             /* in 14 columns    */
                             col=0;
                             printf("\n");}
                     else{col++;}
                     }
                 _setcursortype(_NORMALCURSOR); /* Ret normal cursor*/
```

```
            fclose(fp);                     /* Close file       */
            backout();
                                            /* Leave board in   */
        exit(0);                            /* known state      */
}
```

```
/*********************************************************************/
/*
/* File: HOSTAPP2.CPP Source code for host application
/*
/*********************************************************************/
#include <HI54X.H>
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>

extern int  datareg[],statreg[],ctrlreg[];
extern int  pport, portmode,Readdelay;

void main(void)
{
      FILE *fp;
      if ((fp=fopen("data2.dat","w"))==NULL)       /* Open file  */
      {
            clrscr();
            printf("Cannot open file .\n");
            exit(0);
      }
            portmode=0;                    /* 4-bit mode */
            Readdelay = 20;                /* In case host slow*/
            clrscr();                      /* Clear the screen */
            if((pport=locate_port()) >= 5){/* Find the port.   */
                     printf("No connection\n"); /* If no connection */
                     backout();                /* then leave board */
                     exit(0);}                 /* in known state   */
            else{}
            _setcursortype(_NOCURSOR);         /* Hide text cursor */
            set_latch(1,1);                    /* Keep DSP running */
            int word =0, col=0;                /* and bring PAL out*/
                                               /* out of Tri-state */
                  col=0;
                  gotoxy(1,1);                 /* go to home       */
                  send_word(0x0808, C_SEND);   /* Clear the HINT   */
                  HINT(10000);                 /* Wait for nxt HINT*/
                  send_word(0x1200, A_SEND);   /* Goto 0x46 entries*/
                                               /* before buffer    */
                  for(int buf=0 ; buf < 0x200; buf++)
                                    /* Change here if fs is changed*/
                    {
                     word = read_word(D_READ);  /* Read word from pp*/
                     printf("%4.4x ", word);    /* Print it to scr  */
                     fprintf(fp, "%d\n", word); /* Output to file   */
                     if(col >= 13){             /* in 14 columns    */
                             col=0;
                             printf("\n");}
                     else{col++;}
                     }
```

```
        _setcursortype(_NORMALCURSOR);      /* Ret normal cursor*/
            fclose(fp);                     /* Close file       */
            backout();                      /* Leave board in   */
        exit(0);                           /* known state      */
}
```

```
/*********************************************************************/
/*
/* File: MAIN.C Source code for main program
/*
/*********************************************************************/
#include "math.h"
#include "stddef.h"
#include "stdlib.h"
#include "stdio.h"
#include "conio.h"

#define fs1 625
#define fs2 324

main()
{
        FILE *fp1, *fp2;
        int d1, d2, x1[fs1], x2[fs2];
        int k, out, freq;
        int glde(int k);

        double xro1[fs1], xio1[fs1], xo1[fs1];
        float pi = 3.1415926, tpi;
        int n, u;
        int i, dft1, n1;
        double max1, max2;

        double xro2[fs2], xio2[fs2], xo2[fs2];
        int j, dft2, n2;

        if ((fp1=fopen("data5.dat","rt"))==NULL)       /* Open file  */
        {
                clrscr();
                printf("Cannot open file .\n");
                exit(0);
        }

        if ((fp2=fopen("data6.dat","rt"))==NULL)       /* Open file  */
        {
                clrscr();
                printf("Cannot open file .\n");
                exit(0);
        }
        for(n=0;n<fs1;n++)
        {
                        fscanf(fp1, "%e ", &d1);
                  x1[n]=d1;
        }
        for(n=0;n<fs2;n++)
        {
            fscanf(fp2, "%e ", &d2);
```

```
    x2[n]=d2;
}

fclose(fp1);
fclose(fp2);

tpi=2*pi;
for(u=0;u<fs1;u++)
{
        xro1[u]=0.0;
        xio1[u]=0.0;
        for(n=0;n<fs1;n++)
        {

        /*-- Xr[u] = (1/fs1) sum {xr[n].cos(2PI.u.n/fs1)} --*/
                xro1[u] = xro1[u] + x1[n]*cos(tpi*u*n/fs1);
        /*-- Xi[u] = - (1/fs1) sum xr[n].sin(2PI.u.n/fs1) --*/
                xio1[u] = xio1[u] - x1[n]*sin(tpi*u*n/fs1);
        }
        xro1[u]=xro1[u]/fs1;
        xio1[u]=xio1[u]/fs1;
        xo1[u]=sqrt(xro1[u]*xro1[u]+xio1[u]*xio1[u]);
}
dft1=0;
n1=fs1/2+1;
max1=xo1[0];
for (i=1;i<n1;i++)
{
        if(xo1[i] > max1)
        {
                dft1=i;
                max1=xo1[i];
        }
}

for(u=0;u<fs2;u++)
{
        xro2[u]=0.0;
        xio2[u]=0.0;
        for(n=0;n<fs2;n++)
        {
                xro2[u] = xro2[u] + x2[n]*cos(tpi*u*n/fs2);
                xio2[u] = xio2[u] - x2[n]*sin(tpi*u*n/fs2);
        }
        xro2[u]=xro2[u]/fs2;
        xio2[u]=xio2[u]/fs2;
        xo2[u]=sqrt(xro2[u]*xro2[u]+xio2[u]*xio2[u]);
}

dft2=0;
n2=fs2/2;
```

```
        max2=xo2[0];
        for(j=1;j<n2;j++)
        {
                if(xo2[j] > max2)
                {
                        dft2=j;
                        max2=xo2[j];
                }
        }

        k=dft2-dft1;
        out=glde(k);
        freq=abs(dft1+out*fs1);
        printf("The frequency is %d", freq);
        for(;;);
}

glde(k)
{
        float mult,b1,b2,mtest,md1,md2,mx;

/* This section solves the linear diophantine equation fs1*b1 + fs2*b2 =
na where fs1 and fs2 are the sampling frequencies
and na is the greatest common divisor and returns the value b1, b2 and
na. fs1 and fs2 are assumed positive */

        float bo1,bo2,ma,na,irem,bo3,bo4;
        int iquot;

        bo1=1;
        bo2=0;
        b1=0;
        b2=1;

/* Place fs1 and fs2 in ma(dividend) and na (divisor) respectively */

        ma=fs1;
        na=fs2;

/* Calculate quotient and remainder */

        iquot=ma/na;
        irem=ma-na*iquot;

/* If remainder is not zero, reset dividend and divisor */

        while (irem>0)
        {
                bo3=bo1-iquot*b1;   /* calculate new coefficients */
                bo4=bo2-iquot*b2;
                bo1=b1;             /* redefine bo1, bo2, b1 and b2 */
```

```
                bo2=b2;
                b1=bo3;
                b2=bo4;
                ma=na;              /* redefine dividend and divisor */
                na=irem;
                iquot=ma/na;        /* reapply Euclidean algorithm */
                irem=ma-na*iquot;
        }

/* To check whether the equation is solvable. na must be a factor of k
for the equation to be solvable. */

        mult=k/na;
        if ((k-mult*na)==0)     /* Equation is solvable */
        {
                b1=b1*mult;     /* These new values solve the          */
                                /* diophantine equation                */
                b2=b2*mult;
                mtest=b1;       /* To check whether b1 and b2          */
                                /* are the least values that           */
                md1=fs1/na;     /* satisfies the diophantine equation */
                md2=fs2/na;
                mx=b1;
                mx=mx+md2;

                while ((abs(mx)-abs(b1)) < 0)
                {
                        b1=mx;
                        b2=b2-md1;
                        mx=mx+md2;
                }

                if ((mtest-b1)==0)
                {
                        mx=b1;
                        mx=mx-md2;
                        while ((abs(mx)-abs(b1))<0)
                        {
                                b1=mx;
                                b2=b2+md1;
                                mx=mx-md2;
                        }
                }
        }
return ((int)b1);
}
```

INITIAL DISTRIBUTION LIST

No of copies

1. Defense Technical Information Center 2
   8725 John J. Kingman Rd., Ste 0944
   Ft. Belvoir, VA 22060-6218

2. Dudley Knox Library 2
   Naval Postgraduate School
   411 Dyer Rd.
   Monterey, CA 93943-5101

3. Chairman, Code EC 1
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, CA 93943-5121

4. Professor Phillip E. Pace, Code EC/PC 2
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, CA 93943-5121

5. Professor Curtis Schleher, Code IW/SC 1
   Department of Information Warfare
   Naval Postgraduate School
   Monterey, CA 93943-5121

6. Head, Department of Strategic Studies 1
   SAFTI Military Institute
   Ministry of Defense
   500 Upper Jurong Road
   S638364
   Singapore

7. Maj Chia Eng Seng 2
   Ministry of Defense
   303 Gombak Drive
   S669645
   Singapore